



PLC本体用周辺ツール
CPUユニット

OMRON

プログラマブルコントローラ

SYSMAC

CX-Programmer

Ver.9.□

形CXONE-AL□□D-V4

**オペレーションマニュアル
ファンクションブロック編/
ストラクチャードテキスト編**

— おことわり —

- (1) 本マニュアルの内容の一部または全部を無断で複写、複製、転載することを禁じます。
- (2) 本マニュアルの内容に関しては、改良のため予告なしに仕様などを変更することがありますので、予めご了承ください。
- (3) 本マニュアルの内容に関しては万全を期しておりますが、万一ご不審な点や誤りなどお気付きのことがありましたら、お手数ですが巻末記載の当社支店または営業所までご連絡ください。
その際、巻末記載のマニュアルNo.も併せてお知らせください。

— 商標 —

- ・ Microsoft、Windows、Windows Vistaは、米国Microsoft Corporationの米国およびその他の国における登録商標または商標です。
 - ・ ODVA、CIP、CompoNet、DeviceNet、EtherNet/IPはODVAの商標です。
- その他、本文中に掲載しているシステム名および製品名は、それぞれ各社の商標または登録商標です。

— 著作権 —

スクリーンショットはマイクロソフトの許可を得て使用しています。

CX-Programmer

Ver.9.□

形 CXONE-AL□□D-V4

オペレーションマニュアル
ファンクションブロック編/
ストラクチャードテキスト編

はじめに

このたびは、CX-Programmer をお買い上げいただきまして、ありがとうございます。

本書は、CX-Programmer で使用できるファンクションブロックおよびストラクチャードテキスト言語機能について、関連する仕様および使用方法を記載しています。

CX-Programmer のファンクションブロックおよびストラクチャードテキスト言語機能は、CJ2 CPU ユニット、CS1-H/CJ1-H/CJ1M CPU ユニット Ver.3.0 以降、CS1D CPU ユニット Ver.4.0 以降、CP シリーズ CPU ユニット、NSJ シリーズおよび FQM1 シリーズでサポートしています。

（一部のファンクションブロックおよびストラクチャードテキスト言語機能は、CJ2 CPU ユニット、CS1-H/CJ1-H/CJ1M/CS1D CPU ユニット Ver.4.0 以降のみサポートしています。

詳細については、「バージョンアップのご案内」(P16) を参照してください。)

その他の内容（ファンクションブロックおよびストラクチャードテキスト言語機能以外）に関しては、以下のマニュアルを参照してください。

- ・ CX-Programmer について
 - ： CX-Programmer のオペレーションマニュアル (SBCA-337) およびオペレーションマニュアル SFC 編 (SBCA-348)
- ・ PLC 本体機能について
 - ： CS/CJ/CP/NSJ シリーズの各マニュアル

「プログラマブルコントローラ」の表記について

本マニュアルおよび本ソフトウェアでは、「プログラマブルコントローラ」の略称を「PLC」としています。

ただし、他のプログラマブルコントローラ関連および周辺機器のマニュアルでは「PC」を使用している場合があります。

なお、パソコンは略称を用いず、「パソコン」と表示します。

対象となる読者の方々

本マニュアルは、次の方を対象に記述しています。

電気の知識（電気工事士あるいは同等の知識）および計装の知識を有する方で

- ・ FA 機器の導入を担当される方
- ・ FA システムを設計される方
- ・ FA 現場を管理される方

お願い

- ・ 本マニュアルは、CX-Programmer および CJ2 CPU ユニット、CS1-H/ CJ1-H/CJ1M CPU ユニット ユニット Ver.4.0 以降、CP1H/CP1L CPU ユニット、および NSJ シリーズを使用する上で、必要な情報を記載しています。お使いになる前に本マニュアルをよく読んで、十分に理解してください。また、お読みになった後も本マニュアルは大切に保管して、いつも手元においてお使いください。

「ご承諾事項」

● 保証内容

(1)保証期間

本ソフトウェアの保証期間は、ご購入後またはご指定の場所に納入後1年といたします。ただし「カタログ等」に別途記載がある場合を除きます。

(2)保証範囲

- ・ 本ソフトウェアの使用許諾を受けたお客様が、上記保証期間中にコンピュータ・プログラムの瑕疵（マニュアルとの重要な不一致）を発見し、当社に返却した場合は、当社は瑕疵（当該マニュアルとの重要な不一致）のないコンピュータ・プログラムを記録した媒体と交換いたします。もしくは、当社の選択により、当社ホームページより瑕疵（当該マニュアルとの重要な不一致）のないコンピュータ・プログラムをダウンロードしていただく方法により提供いたします。また、当社の責任によるコンピュータ・プログラムの記録媒体の不良を発見し、当社に返却した場合、当社は、無償で、良品の媒体に記録したコンピュータ・プログラムと交換いたします。
- ・ 万一、当社がコンピュータ・プログラムの瑕疵（マニュアルとの重要な不一致）を除去できないと判断した場合は、お客様が本ソフトウェア購入代金として支払った金額をお返しいたします。

● 責任の制限

- ・ 前条に定める交換または購入代金の返金は、本ソフトウェアの保証責任のすべてを定めるものであり、当社は本ソフトウェアの瑕疵により発生した、お客様の直接的、間接的あるいは波及効果による損害等いかなる損害に対しても一切の責任を負いません。
- ・ 当社は、本ソフトウェアを当社以外の第三者が変更、改造することにより生じた瑕疵や、本ソフトウェアのインストールその他の使用によって当社商品、インストールされたソフトウェア、またはすべてのコンピュータ機器、コンピュータプログラム、ネットワーク、データベースがDDoS攻撃（分散型DoS攻撃）、コンピュータウイルスその他の技術的な有害プログラム、不正アクセスを受けたことにより生じた損害につきましては、一切責任を負いません。
- ・ 当社は、本ソフトウェアに基づき、当社以外の第三者が開発したソフトウェアおよびそれにより生じた結果について一切の責任を負いません。

● 本ソフトウェアの用途

本ソフトウェアを本マニュアルに記載の用途以外の用途で使用しないでください。

● 仕様の変更

本ソフトウェアの仕様および付属品は改善またはその他の事由により、必要に応じて、変更される場合があります。

● 適用範囲

以上の内容は、日本国内での取引および使用を前提としております。日本国外での取引および使用に関しては、当社営業担当者までご相談ください。

関連マニュアル

CX-Programmer Ver.9.□の関連マニュアル

CX-Programmer Ver.9.□の関連マニュアルは、下表のように構成されています。併せてご覧ください。

Man.No.	形式	マニュアル名称	用途	内容
本書 (SBCA-338)	形 CXONE-AL□□D-V4	CX-Programmer オペレーションマニュアルファンクションブロック編/ストラクチャードテキスト編	ファンクションブロック機能およびストラクチャードテキスト言語について知りたいとき	CX-Programmer のファンクションブロック/ストラクチャードテキストの機能と使用方法について、説明しています。その他 CX-Programmer と共通の機能については、No.SBCA-337 を参照してください。
SBCA-337	形 CXONE-AL□□D-V4	CX-Programmer オペレーションマニュアル	ファンクションブロック機能およびストラクチャードテキスト言語以外について知りたいとき	CX-Programmer の操作方法について説明しています。
SBCA-348	形 CXONE-AL□□D-V4	CX-Programmer オペレーションマニュアルSFC編	SFC プログラミング機能について知りたいとき	SFC プログラミング機能について説明しています。その他 CX-Programmer と共通の機能については、No.SBCA-337 を参照してください。
SBCA-347	形 CXONE-AL□□D-V4	CX-Integrator CS/CJ シリーズ用ネットワーク コンフィグレーションツール オペレーションマニュアル	ネットワーク設定/モニタを行いたいとき	CX-Integrator の操作方法について説明しています。
SBCA-346	形 CXONE-AL□□D-V4 形 CXONE-LT□□□-V4	CX-One セットアップマニュアル	FA 統合ツールパッケージ CX-One からセットアップするとき	FA 統合ツールパッケージ CX-One の概要、CX-One のインストール方法を記載しています。

CJ2H/CJ2M/ CS1-H/ CJ1-H/CJ1M CPU ユニットの関連マニュアル

本マニュアルに記載してない CJ2H/ CJ2M/CS1-H/ CJ1-H/CJ1M CPU ユニットの仕様/取扱いについては、CS/CJ シリーズの以下の各マニュアルを参照してください。

Man.No.	形式	マニュアル名称	用途	内容
SBCA-349	形 CJ2H-CPU6□-EIP 形 CJ2H-CPU6□ 形 CJ2M-CPU1□ 形 CJ2M-CPU3□	CJ シリーズ CJ2 CPU ユニ ット ユーザーズ ハードウェア編	CJ CPU ユニ ットのハー ドウェア的 な仕様につ いて知りたい とき	CJ2 CPU ユニットに関して、以下の内容を説明しています。 ・概要／特長を知りたい ・基本システムの構成を知りたい ・各部の名称と機能を知りたい ・取り付けと設定方法を知りたい ・トラブル時の対処方法を知りたい ユーザーズマニュアル ソフトウェア編 (No.SBCA-350) と併せて使用してください。
SBCA-350	形 CJ2H-CPU6□-EIP 形 CJ2H-CPU6□ 形 CJ2M-CPU1□ 形 CJ2M-CPU3□	CJ シリーズ CJ2 CPU ユニ ット ユーザーズ マニュアル ソフトウェア編	CJ CPU ユニ ットのソフト ウェア的 な仕様につ いて知りたい とき	CJ2 CPU ユニットに関して、以下の内容を説明しています。 ・CPU ユニットの動作について知りたい ・内部メモリについて知りたい ・プログラムについて知りたい ・各種設定について知りたい ・CPU 内蔵機能について知りたい ユーザーズマニュアル ハードウェア編 (No.SBCA-349) と併せて使用してください。
SBCA-301	形 CS1G/H-CPU□□H	CS シリーズ ユーザーズマニ ュアル セットアップ編	CS シリーズ の概要／設 計／取付／ 保守などの 基本的な仕 様について 知りたいとき	CS シリーズの PLC 本体に関して、以下の内容を説明しています。 ・概要／特長を知りたい ・システム構成を設計したい ・取付／配線をしたい ・I/O メモリの割付を知りたい ・トラブル時の対処方法を知りたい ユーザーズマニュアル プログラミング編 (No.SBCA-313) と併せて使用してください。
SBCA-312	形 CJ1H-CPU□□H-R 形 CJ1G/H-CPU□□H 形 CJ1M-CPU□□ 形 CJ1G-CPU□□P 形 CJ1G-CPU□□	CJ シリーズ ユーザーズマニ ュアル セットアップ編	CJ シリーズ の概要／設 計／取付／ 保守などの 基本的な仕 様について 知りたいとき	CJ シリーズの PLC 本体に関して、以下の内容を説明しています。 ・概要／特長を知りたい ・システム構成を設計したい ・取付／配線をしたい ・I/O メモリの割付を知りたい ・トラブル時の対処方法を知りたい ユーザーズマニュアル プログラミング編 (No.SBCA-313) と併せて使用してください。

Man.No.	形式	マニュアル名称	用途	内容
SBCA-313	形 CS1G/H-CPU□□H 形 CS1G/H-CPU□□-V1 形 CS1D-CPU□□HA 形 CS1D-CPU□□SA 形 CS1D-CPU□□P 形 CS1D-CPU□□H 形 CS1D-CPU□□S 形 CJ1H-CPU□□H-R 形 CJ1G/H-CPU□□H 形 CJ1G-CPU□□P 形 CJ1M-CPU□□ 形 CJ1G-CPU□□ 形 NSJ□-□□□□(B)-G5D 形 NSJ□-□□□□(B)-M3D	CS/CJ/NSJ シリーズ ユーザーズマニュアル プログラミング編	CS/CJ/NSJ シリーズの各種機能について知りたいとき	CS/CJ/NSJ シリーズの PLC 本体に関して、以下の内容を説明しています。 ・プログラミングをしたい ・タスク機能を知りたい ・ファイルメモリ機能を知りたい ・各種の機能を知りたい ユーザーズマニュアル セットアップ編 (CS シリーズ: No. SBCA-301、CJ シリーズ: No. SBCA-312) と併せて使用してください。
SBCA-351	形 CJ2H-CPU6□-EIP 形 CJ2H-CPU6□ 形 CJ2M-CPU□□ 形 CS1G/H-CPU□□H 形 CS1G/H-CPU□□-V1 形 CS1D-CPU□□HA 形 CS1D-CPU□□SA 形 CS1D-CPU□□P 形 CS1D-CPU□□H 形 CS1D-CPU□□S 形 CJ1H-CPU□□H-R 形 CJ1G/H-CPU□□H 形 CJ1G-CPU□□P 形 CJ1M-CPU□□ 形 CJ1G-CPU□□ 形 NSJ□-□□□□(B)-G5D 形 NSJ□-□□□□(B)-M3D	CS/CJ/NSJ シリーズ コマンドリファレンス	命令語の詳細について知りたいとき	各命令語の詳細説明をしています。 プログラミング時に、ユーザーズマニュアル セットアップ編 (CS シリーズ: No.SBCA-301、CJ シリーズ: No.SBCA-312)、ユーザーズマニュアル プログラミング編 (No.SBCA-313) と併せて使用してください。 CJ2 CPU ユニットを使用する場合は、CJ2 CPU ユニットユーザーズマニュアル ハードウェア編 (No.SBCA-349) およびソフトウェア編 (No.SBCA-350) を併せて使用してください。
SBCA-304	形 CS1G/H-CPU□□H 形 CS1G/H-CPU□□-V1 形 CS1D-CPU□□HA 形 CS1D-CPU□□SA 形 CS1D-CPU□□P 形 CS1D-CPU□□H 形 CS1D-CPU□□S 形 CS1W-SCU□□-V1 形 CS1W-SCB□□-V1 形 CJ1H-CPU□□H-R 形 CJ1G/H-CPU□□H 形 CJ1G-CPU□□P 形 CJ1M-CPU□□ 形 CJ1G-CPU□□ 形 CJ2H-CPU6□-EIP 形 CJ2H-CPU6□ 形 CJ2M-CPU□□ 形 CJ1W-SCU□□-V1 形 CP1H-X□□□□-□ 形 CP1H-XA□□□□-□ 形 CP1H-Y□□□□-□ 形 CP1L-M/L□□□□-□ 形 CP1E-E□□□D□-□ 形 CP1E-N□□□D□-□	CS/CJ シリーズ 通信コマンドリファレンス	CS/CJ シリーズ CPU ユニット宛て通信コマンドの詳細について知りたいとき	1)C モードコマンドおよび 2)FINS コマンドの詳細について説明しています。 CPU ユニット宛ての通信コマンド (C モードコマンドまたは FINS コマンド) の詳細を知りたいときに、参照してください。 注: 本マニュアルに記載している通信コマンドは、CPU ユニット宛ての通信コマンドです。その通信経路は、問いません (CPU ユニットのシリアル通信ポート、シリアルコミュニケーションボード/ユニットの通信ポート、通信ユニット経由などが可能です)。

NSJ シリーズの関連マニュアル

本マニュアルに記載していない NSJ シリーズの仕様/取扱いについては、NSJ シリーズの以下のマニュアルを参照してください。

Man.No.	形式	マニュアル名称	用途	内容
SBCA-342	形 NSJ5-TQ□□(B)-G5D 形 NSJ5-SQ□□(B)-G5D 形 NSJ8-TV□□(B)-G5D 形 NSJ10-TV□□(B)-G5D 形 NSJ12-TS□□(B)-G5D 形 NSJ5-TQ□□(B)-M3D 形 NSJ5-SQ□□(B)-M3D 形 NSJ8-TV□□(B)-M3D	NSJ シリーズ NSJ コントローラ ユーザーズマニュアル セットアップ編	NSJ シリーズの概要/設計/取付/保守などの基本的な仕様について知りたいとき	NSJ シリーズに関して、以下の内容を説明しています。 ・概要/特長を知りたい ・システム構成を設計したい ・取付/配線をしたい ・I/O メモリの割付を知りたい ・トラブル時の対処方法を知りたい CS/CJ シリーズ ユーザーズマニュアル セットアップ編 (SBCA-312) /プログラミング編 (SBCA-313) /NS-V1/-V2 シリーズ セットアップマニュアル (SBSA-554) と併せて使用してください。

FQM1 シリーズ ユニット Ver.3.0 以降の関連マニュアル

本マニュアルに記載していない FQM1 シリーズ ユニット Ver.3.0 以降（形 FQM1-CM002/MMP22/MMA22）の仕様/取扱いについては、FQM1 シリーズユニット Ver.3.0 の以下のマニュアルを参照してください。

Man.No.	形式	マニュアル名称	用途	内容
SBCE-338	形 FQM1-CM002 形 FQM1-MMP22 形 FQM1-MMA22	FQM1 シリーズ ユーザーズマニュアル	FQM1 シリーズの 概要/設計/取付 /保守などの基本的な仕様について 知りたいとき	FQM1 シリーズ ユニット Ver.3.0 の 各モジュールに関して、以下の内容を説明しています。 ・概要/特長を知りたい ・システム構成を設計したい ・取付/配線をしたい ・I/O メモリの割付を知りたい ・トラブル時の対処方法を知りたい
SBCE-339	形 FQM1-CM002 形 FQM1-MMP22 形 FQM1-MMA22	FQM1 シリーズ コマンドリファレンスマニュアル	命令語の詳細について知りたいとき	各命令語の詳細説明をしています。 プログラミング時に、ユーザーズマニュアル（SBCE-338）と併せて使用してください。

CP シリーズの関連マニュアル

本マニュアルに記載していない CP シリーズの仕様/取扱いについては、CP シリーズの以下のマニュアルを参照してください。

Man.No.	形式	マニュアル名称	用途	内容
SBCA-340	形 CP1H-X□□□□-□ 形 CP1H-XA□□□□-□ 形 CP1H-Y□□□□-□	CP シリーズ CPIH CPU ユニット ユーザーズマニュアル	CP シリーズ CPIH の概要/設計/取付/保守などの基本的な仕様について知りたいとき	CP シリーズ CPIH の PLC 本体に関して、以下の内容を説明しています。 ・概要/特長を知りたい ・システム構成を設計したい ・取付/配線をしたい ・I/O メモリの割付を知りたい ・トラブル時の対処方法を知りたい プログラミングマニュアル（SBCA-341）と併せて使用してください。
SBCA-345	形 CP1L-M□□□□-□ 形 CP1L-L□□□□-□	CP シリーズ CPIL CPU ユニット ユーザーズマニュアル	CP シリーズ CPIL の概要/設計/取付/保守などの基本的な仕様について知りたいとき	CP シリーズ CPIL の PLC 本体に関して、以下の内容を説明しています。 ・概要/特長を知りたい ・システム構成を設計したい ・取付/配線をしたい ・I/O メモリの割付を知りたい ・トラブル時の対処方法を知りたい プログラミングマニュアル（SBCA-341）と併せて使用してください。
SBCA-341	形 CP1H-X□□□□-□ 形 CP1H-XA□□□□-□ 形 CP1H-Y□□□□-□ 形 CP1L-M□□□□-□ 形 CP1L-L□□□□-□	CP シリーズ CPIH/CPIL CPU ユニット プログラミングマニュアル	CP シリーズ CPIH/CPIL のプログラミングについて知りたいとき	CP シリーズ CPIH/CPIL の PLC 本体に関して、以下の内容を説明しています。 ・プログラミングをしたい ・タスク機能を知りたい ・各命令語の詳細を知りたい。

Man.No.	形式	マニュアル名称	用途	内容
SBCA-477	形 CP2E-E□□D□-□ 形 CP2E-S□□D□-□ 形 CP2E-N□□D□-□	SYSMAC CP シリーズ CP2E CPU ユニットの ユーザーズマニュアル ハードウェア編	CP2E のハードウェア的な仕様について知りたいとき	CP2E の PLC 本体に関して、以下の内容を説明しています。 ・概要／特長 ・基本システムの構成 ・各部の名称と機能 ・取り付けと設定方法 ・トラブル時の対処方法
SBCA-478	形 CP2E-E□□D□-□ 形 CP2E-S□□D□-□ 形 CP2E-N□□D□-□	SYSMAC CP シリーズ CP2E CPU ユニットの ユーザーズマニュアル ソフトウェア編	CP2E のソフトウェア的な仕様について知りたいとき	CP2E の PLC 本体に関して、以下の内容を説明しています。 ・CPU ユニットの動作 ・内部メモリ ・プログラム ・各種設定 ・CPU 内蔵機能 ・割込機能 ・高速カウンタ入力機能 ・パルス出力機能 ・シリアル通信機能 ・Ethernet 通信機能 ・アナログオプションボード機能 ・その他機能
SBCA-356	形 CP1E-E□□D□-□ 形 CP1E-N□□D□-□ 形 CP1E-NA□□D□-□ 形 CP2E-E□□D□-□ 形 CP2E-S□□D□-□ 形 CP2E-N□□D□-□	SYSMAC CP シリーズ CP1E/CP2E CPU ユニットの コマンドリファレンスマニュアル	命令語の詳細について知りたいとき	各命令語の詳細説明をしています。

安全上のご注意

安全に使用していただくための表示と意味について

このユーザーズマニュアルでは、CX-Programmer およびプログラマブルコントローラ（PLC）を安全に使用していただくために、注意事項を次のような表示と図記号で示しています。

ここで示した注意事項は、安全に関する重大な内容を記載しています。必ず守ってください。

表示と意味は次のとおりです。



警告

正しい取扱いをしなければ、この危険のために、軽傷・中程度の傷害を負ったり万一の場合には重傷や死亡に至る恐れがあります。また、同様に重大な物的損害をもたらす恐れがあります。



注意

正しい取扱いをしなければ、この危険のために、時に軽傷・中程度の傷害を負ったり、あるいは物的損害を受ける恐れがあります。

安全上の要点

製品を安全に使用するために実施または回避すべきことを示します。

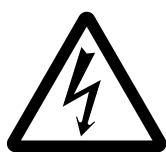
使用上の注意

製品が動作不能、誤動作、または性能・機能への悪影響を予防するために実施または回避すべきことを示します。

お願い

本文中の「お願い」は安全上の要点、使用上の注意と同等の内容を示します。

図記号の説明



△記号は、注意（警告を含む）を意味しています。
具体的な内容は、△の中と文章で示します。
左図の場合は、「感電注意」を表します。



⊘記号は、禁止を意味しています。
具体的な内容は、⊘の中と文章で示します。
左図の場合は、「分解禁止」を表します。



●記号は、強制を意味しています。
具体的な内容は、●の中と文章で示します。
左図の場合は、「一般的な強制事項」を表します。



△記号は、注意（警告を含む）を意味しています。
具体的な内容は、△の中と文章で示します。
左図の場合は、「一般的な注意」を表します。

警告

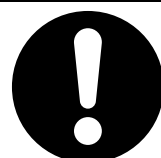
CX-Programmer を使用して、I/O メモリエリアの状態を実機 PLC へ転送するときは、十分に安全を確認してから行ってください。
CPU ユニットの動作モードにかかわらず、出力ユニットに接続された機器が誤動作する恐れがあります。



CX-Programmer を使用して、**CPU** ユニットの I/O メモリエリアにデータを転送する場合、注意すべき機能は、以下のとおりです。

- CX-Programmer** から **CPU** ユニットの実 I/O (CIO エリア) への転送 (PLC メモリウィンドウ機能による)
- ファイルメモリから **CPU** ユニットの実 I/O (CIO エリア) への転送 (メモ리카ードウィンドウ機能による)

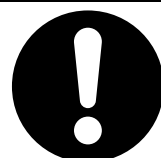
本ソフトウェアを使用するパソコンにウィルス対策ソフトを導入し、パソコンがウィルスに感染するリスクを低減させてください。また、ウィルス対策ソフトは常に最新の状態にしてください。



OS の脆弱性に起因するセキュリティリスクを回避するために、本ソフトウェアを使用するパソコンの **OS** を常に最新の状態にしてください。



機能の追加や操作性の改善、セキュリティの強化のため、本ソフトウェアは、常に最新のバージョンにアップデートしてお使いください。



本ソフトウェアで設定するユーザ名やパスワードについて、他人から不正に利用されないよう適切な設定・管理をしてください。



制御システムや装置のネットワークはファイアウォール (未使用通信ポートの遮断、通信ホストの制限) を導入し、IT ネットワークからの分離を行い、本ソフトウェアの制御システムへの接続はファイアウォール内で行うようにしてください。



本ソフトウェアから制御システムや装置へのリモートアクセスが必要な場合は仮想プライベートネットワーク (VPN) を使用してください。



⚠ 注意

ファンクションブロックで、命令のオペランドが、複数ワード (CH) の先頭アドレス (または最終アドレス) を指定している場合、必ず、AT 指定の変数 (または外部変数指定) か、または命令が処理するデータサイズと同一のサイズの変数を、指定してください。以下の点にご注意ください。



1) データサイズが一致していない非配列型変数を AT 指定せずに指定した場合

CX-Programmer がコンパイル時にエラーを出力します。

2) 配列型の変数を指定した場合

●命令が処理をするサイズが固定のオペランドの場合：

配列の要素数は、必ず、命令が処理をするサイズと同一の要素数を、確保してください。さもないと、**CX-Programmer** がコンパイル時にエラーを出力します。

●命令が処理をするサイズが可変のオペランドの場合：

必ず、配列の要素数 $\geq$ 他のオペランドで指定したサイズを確保してください。

・サイズを指定する他のオペランドが、定数の場合：

CX-Programmer Ver.5.0 以降は、コンパイル時にエラーを出力します。

・サイズを指定する他のオペランドが、変数の場合：

配列の要素数が、他のオペランド (変数) で指定したサイズと不一致でも、**CX-Programmer Ver.5.0** 以降は、コンパイル時にエラーを出力しません (ワーニングメッセージは表示します)。

とくに、配列の要素数 $<$ 他のオペランドで指定したサイズの場合

(例：命令が処理をするサイズ=16 で、実際に変数テーブル上で登録した要素数=10 の場合)、命令は、その要素数を越えたエリアに対しても、読み書きなどの処理を実行します (例：実際に変数テーブル上で登録した要素数の、次のアドレスの 6CH 分に対しても、読み書きなどの処理をします)。他の命令で、そのエリアを使用していた場合 (変数の内部割り当てを含む)、思わぬ動作をし、重大な事故につながる恐れがあります。

「オペランドに指定している変数のサイズ」 $<$ 「オペランド定義サイズ」となってもシステムに影響がないことを確認した上で、PLC の運転を開始するようにしてください。

他ノードへプログラムを転送するときや、I/O メモリを変更する操作をするときは、変更先ノードの安全を確認してから行ってください。

けがをする恐れがあります。



サイクルタイムが延びても影響がないことを確認してから、オンラインエディットをしてください。
入力信号を読み取れないことがあります。



ユニット間同期制御機能使用時、同期処理時間が延びても主軸と従軸の動作に影響がないことを確認してからオンラインエディットをしてください。



ラダーセクションウィンドウで導通/現在値モニタをする場合、またはウォッチウィンドウにて現在値モニタをする場合、十分な安全を確認してから行ってください。
ショートカットキーの誤操作によって、強制セット/リセット、またはセット/リセットを行うと、**CPU** ユニットの動作モードにかかわらず、出力ユニットに接続された機器が誤動作する恐れがあります。



安全上の要点

CX-Programmer を使用する際には、次のことに気を付けてください。

- ・ CX-Programmer を起動する前に、以下の点にご注意ください。
 - ・ CX-Programmer の使用に直接関係がない他のアプリケーションは、すべて終了してください。とくに、スクリーンセーバ、ウイルスチェック、電子メールなどの通信ソフト、スケジュール管理ソフトなど、定期/不定期に起動するアプリケーションは、その機能を必ず停止させてください。
 - ・ パソコンのハードディスクやプリンタをネットワークで他のパソコンに共有させている時は、共有を解除してください。
 - ・ ノートパソコンで、初期設定が RS-232C ポートをモデムや赤外線に割当てているものがあります。パソコンの説明書にしたがって RS-232C ポートを通常のシリアルポートとして使用できるように設定を変更してください。
 - ・ ノートパソコンで、初期設定が省電力のために RS-232C、USB および Ethernet ポートに規定の電力を供給しない設定になっているものがあります。省電力設定は Windows の設定ほか、パソコン独自のユーティリティや BIOS で設定されている場合があります。パソコンの説明書にしたがって、すべての省電力機能を停止させてください。
- ・ CX-Programmer と PLC をオンライン接続中は、PLC の電源を切ったり接続ケーブルの抜き差しをしないでください。パソコンが誤動作する場合があります。
- ・ 設備に影響がないことを確認してから、現在値を変更してください。
- ・ 設備に影響がないことを確認してから、設定値を変更してください。
- ・ 設備に影響がないことを確認してから、CPU ユニットの動作モードを切り替えてください。
- ・ 設備に影響がないことを確認してから、強制セット／強制リセットしてください。
- ・ 作成したユーザプログラムは、十分な動作確認を行った後、本運転に移行してください。
- ・ CJ2/ CS1-H/ CJ1-H/ CJ1M/ CS1D/ CP1/ CP2 CPU ユニットの場合、オンラインエディットを行うと、通常の CPU ユニット内蔵メモリ (RAM) への書き込み後、続けて CPU ユニット内蔵フラッシュメモリに対する書き込み (バックアップ) 処理が行われます。フラッシュメモリへの書き込み (バックアップ) 処理中 (CPU ユニット前面の「BKUP」LED が点灯中) は、PLC 本体の電源を OFF にしないでください。
なお、フラッシュメモリへの書き込みステータスを表示させるには、パソコンのプロパティで「バックアップ中のステータスのダイアログを表示する」をチェックし、かつ [表示] メニューから [ウィンドウ] | [フラッシュバックアップモニタ] を選択します。

- ・ファンクションブロック（ラダー言語または ST 言語）を含むプログラムも、ファンクションブロックを含まない通常のプログラムと同様、ダウンロード／アップロードできます。
ただし、ファンクションブロックを含むタスクを、タスク単位でダウンロードすることはできません（アップロードは可能です）。
- ・CX-Programmer Ver.5.0 以降で作成したファンクションブロックを含むユーザプログラムを、CS/CJ シリーズ CPU ユニット ユニット Ver.2.0 以前にダウンロードすると、すべてのインスタンスは、不正命令となり、ユーザプログラムは編集および実行することはできません。
- ・入力変数および入出力変数のデータ型が **BOOL** 型以外の場合、パラメータに、単に数値のみ（例：20）を入力すると、数値ではなく、CIO エリアの CH アドレス（例：0020CH）の値を渡してしまいますので、ご注意ください。値を設定する場合、単に数値を入力しないで、必ず、&、#、または +、- の接頭辞の後に、数値を入力してください。
- ・入力パラメータにアドレスを設定することは可能ですが、その場合、入力変数にアドレス自体を渡すことはできません（入力パラメータにアドレスを設定しても、入力変数のデータ型サイズ分の値が、ファンクションブロックに渡されます）。
したがって、ファンクションブロック内の命令のオペランドが、複数ワード（CH）の先頭アドレス（または最終アドレス）を指定している場合、そのオペランドを「入力変数」とすることはできません。
その場合、「配列型指定」の「入出力変数」（入力パラメータに先頭アドレスを設定）として配列型変数の先頭（または最終）を指定するか（CX-Programmer Ver.7.0 以降のみ）、「内部変数」として AT 指定するか、または「配列型指定」の「内部変数」として配列型変数の先頭（または最終）を指定してください。
- ・入力パラメータから「入力変数」および「入出力変数」には、（アルゴリズム内命令実行のタイミングでなく）アルゴリズムの前に一括で、値が渡されます。
したがって、ファンクションブロックのアルゴリズム内の命令実行のタイミングで、パラメータから「入力変数」「入出力変数」に値を渡すことが必要な場合は、「入力変数」「入出力変数」としないで、「内部変数」または「外部変数」としてください。
出力変数からパラメータへ値を書き込むタイミングも同様です。
- ・以下の場合、必ず、内部変数として、AT 指定を使用してください。
 - ・基本 I/O ユニットまたは高機能ユニットの割付アドレスで、かつその割付アドレスをグローバル変数へ登録してその変数を外部変数として指定することができない場合（グローバル変数とのデータセットが保証できないときなど）
 - ・あらかじめ外部変数に登録されている以外の特殊補助リレーで、かつその特殊補助リレーをグローバル変数へ登録してその変数を外部変数として指定しない場合
 - ・ネットワーク上の他ノードの、PLC のアドレス指定時：例）SEND 命令の送信先（相手ノード）受信開始 CH 番号、RECV 命令の送信要求元（自ノード）受信開始 CH 番号
 - ・命令のオペランドが、複数ワード（CH）の先頭アドレス（または最終アドレス）を指定している場合で、かつそのオペランドを配列指定することができない場合（例：配列要素数を特定化できないとき）

マニュアル改訂履歴

マニュアル改訂記号は、表表紙・裏表紙の左下に記載されている Man.No. の後尾に付記されます。



マニュアル改訂記号は、表表紙・裏表紙の左下に記載されている Man. No. の後尾に付記されます。

改訂記号	改訂年月	改訂理由・改訂ページ
—	2005 年 2 月	初版
B	2005 年 11 月	CX-Programmer Ver.6.0 から Ver.6.1 へのバージョンアップに伴う改訂
C	2006 年 7 月	CX-Programmer Ver.6.1 から Ver.7.0 へのバージョンアップに伴う改訂
D	2007 年 1 月	誤記修正
E	2007 年 7 月	CX-Programmer Ver.7.0 から Ver.7.2 へのバージョンアップに伴う改訂
G	2008 年 6 月	CX-Programmer Ver.7.2 から Ver.8.0 へのバージョンアップに伴う改訂
H	2009 年 2 月	CX-Programmer Ver.8.0 から Ver.8.1 へのバージョンアップに伴う改訂
J	2009 年 12 月	CX-Programmer Ver.8.3 から Ver.9.0 へのバージョンアップに伴う改訂、誤記修正
K	2010 年 2 月	CX-Programmer Ver.9.0 から Ver.9.1 へのバージョンアップに伴う改訂、誤記修正
L	2010 年 10 月	CX-Programmer Ver.9.1 から Ver.9.2 へのバージョンアップに伴う改訂、誤記修正
M	2011 年 1 月	CX-Programmer Ver.9.2 から Ver.9.3 へのバージョンアップに伴う改訂
N	2012 年 8 月	誤記修正
P	2015 年 10 月	CX-Programmer Ver.9.5 から Ver.9.6 へのバージョンアップに伴う改訂、誤記修正
Q	2016 年 4 月	CX-One 形式変更に伴う改訂
R	2018 年 12 月	CX-Programmer Ver.9.6 から Ver.9.7 へのバージョンアップに伴う改訂
S	2019 年 9 月	CX-Programmer Ver.9.7 から Ver.9.72 へのバージョンアップに伴う改訂
T	2022 年 10 月	セキュリティに関する安全上の注意追記に伴う改訂

バージョンアップのご案内

CX-Programmer ファンクションブロック、およびストラクチャードテキスト以外の強化機能については、CX-Programmer オペレーションマニュアル（SBCA-337）を参照してください。

■ Ver.9.7 から Ver.9.72 への機能強化

ファンクションブロック／ストラクチャードテキストをサポートする新規機種形 CP2E-E□□DR-A、形 CP2E-S□□D□-□、形 CP2E-N□□D□-□が使用可能になりました。

■ Ver.9.6 から Ver.9.7 への機能強化

ファンクションブロック／ストラクチャードテキストをサポートする新規機種形 CS1D-CPU□□HA、形 CS1D-CPU□□SA が使用可能になりました。

■ Ver.9.5 から Ver.9.6 への機能強化

[対応 PLC 機種]

以下 CPU ユニットバージョンをサポートしました。拡張型パスワードによるリードプロテクト機能が利用可能です。

CJ2H CPU ユニット Ver.1.5, CJ2M CPU ユニット Ver.2.1, CS1G-H CPU ユニット Ver.4.1, CS1H-H CPU ユニット Ver.4.1, CJ1G-H CPU ユニット Ver.4.1, CS1D-H CPU ユニット Ver.1.4, CS1D-S CPU ユニット Ver.2.1, CP1H CPU ユニット Ver.1.3, CP1L CPU ユニット Ver.1.1, CP1L-L CPU ユニット Ver.1.1, CP1E CPU ユニット Ver.1.3

[パスワードによるリードプロテクト機能]

拡張型パスワードによるリードプロテクト機能を追加しました。
読出プロテクトのパスワードを拡張し、ユーザの設計資産を強固に保護します。

■ Ver.9.2 から Ver.9.3 への機能強化

[ST エディタビューの強化]

ST エディタビューにおいて、以下の改善を実施しました。

- ・行番号の表示が追加されました。また、行番号を指定したジャンプが可能になりました。
- ・関数および登録済変数のリスト選択が可能になりました。
- ・制御構文フォーマットをポップアップメニューから入力可能になりました。
- ・赤色波線表示およびエラー情報のツールチップ表示により、ST 文法エラー箇所をプログラムチェック前に確認可能になりました。
- ・選択行のコメント化、コメント化解除がメニューおよびツールバー選択により可能になりました。
- ・行のインデントがメニューおよびツールバー選択により可能になりました。

[スマートインプットの強化]

- ・FB ラダービューにおいて、タスクラダービューと同様にスマートインプット入力モードでの入力が可能になりました。

■ Ver.9.1 から Ver.9.2 への機能強化

[構造体サポートの強化]

CJ2CPU ユニットにおいて、構造体の使用可能範囲を拡張しました。

- ・構造体（構造体変数、構造体メンバ変数、構造体配列変数）を、ST（ストラクチャードテキスト）プログラムにおいて使用可能になりました。
- ・構造体変数を、FB（ファンクションブロック）ラダー/ST の外部変数として登録・使用可能になりました。

● 構造体変数の使用可否 — 旧バージョンとの比較表

使用箇所			Ver.9.2 未満	Ver.9.2 以降
グローバル変数テーブル			○	○
ラダープログラム	ローカル変数テーブル	セクションビュー	○	○
		ローカル変数テーブル	×	○
ST プログラム	ST エディタ	ローカル変数テーブル	×	○
		ST エディタ	×	○
SFC プログラム	ローカル変数テーブル	ローカル変数テーブル	×	○
		SFC チャートビュー	×	×
		サブチャートビュー	×	×
		サブチャート変数テーブル	×	○
		アクションラダービュー	×	○
		アクション ST ビュー	×	○
		トランジションラダービュー	×	○
		トランジション ST ビュー	×	○
FB ラダー	変数	内部変数	○	○
		入力変数	×	×
		出力変数	×	×
		入出力変数	○	○
		外部変数	×	○
FBST	変数	内部変数	×	○
		入力変数	×	×
		出力変数	×	×
		入出力変数	×	○
		外部変数	×	○

[TIMER 型/COUNTER 型変数サポートの強化]

CJ2CPU ユニットにおいて、TIMER/COUNTER 型変数の使用可能範囲を拡張しました。

- ・TIMER/COUNTER 型変数を、ST プログラムにおいて使用可能になりました。タイムアップ/カウントアップフラグ、タイマ/カウンタ現在値を ST プログラム内で使用することができます。
- ・ST プログラム内で、タイマ/カウンタの起動と停止が可能になりました。
- ・TIMER/COUNTER 型変数を、FB の外部変数として登録・使用可能になりました。

● TIMER 型/COUNTER 型変数の使用可否 — 旧バージョンとの比較表

使用箇所			Ver.9.2 未満	Ver.9.2 以降
グローバル変数テーブル			○	○
ラダープログラム	ローカル変数テーブル		○	○
	セクションビュー		○	○
ST プログラム	ローカル変数テーブル		○	○
	ST エディタ		×	○
SFC プログラム	ローカル変数テーブル		○	○
	SFC チャートビュー		×	×
	サブチャートビュー		×	×
	サブチャート変数テーブル		○	○
	アクションラダービュー		○	○
	アクション ST ビュー		×	○
	トランジションラダービュー		○	○
	トランジション ST ビュー		×	○
FB ラダー	変数	内部変数	○	○
		入力変数	×	×
		出力変数	×	×
		入出力変数	×	×
		外部変数	×	○
FBST	変数	内部変数	×	○
		入力変数	×	×
		出力変数	×	×
		入出力変数	×	×
		外部変数	×	○

■ Ver.9.0 から Ver.9.1 への機能強化

- ・ファンクションブロック／ストラクチャードテキストをサポートする新規機種形 CJ2M-CPU□□が使用可能になりました。
- ・PLC 機種が「CJ2M」のとき、メモリ表示機能で、FB プログラムエリアの使用量の表示が可能になりました。

■ Ver.8.3 から Ver.9.0 への機能強化

[変数のデータ型として、構造体型をサポート]

従来 (Ver.8.3)	今回 (Ver.9.0)
構造体には未対応。	PLC 機種が CJ2 シリーズの場合、変数のデータ型として、構造体型をサポート。

■ Ver.8.0 から Ver.8.1 への機能強化

ファンクションブロック／ストラクチャードテキストをサポートする新規機種形 CJ2H-CPU6□が使用可能になりました。

■ Ver.7.2 から Ver.8.0 への機能強化

ファンクションブロック／ストラクチャードテキストをサポートする新規機種形 CJ2H-CPU6□-EIP が使用可能になりました。

■ Ver.7.0 から Ver.7.2 への機能強化

[IEC61131-3 言語の強化]

IEC61131-3 言語であるストラクチャードテキスト言語およびSFC言語のサポートを強化しました。

ラダー、ストラクチャードテキスト（以下 ST）、SFC を任意に組み合わせて、処理に応じた最適な言語でユーザプログラムを記述できるため、プログラム開発工数を削減し、プログラムの視認性を向上させることができます。

● プログラム（タスクに割付）の言語として ST 言語をサポート

従来 (Ver.7.0)	今回 (Ver.7.2)
ST 言語は、ファンクションブロック内では使用できませんでした。	ST 言語を、ファンクションブロック外のプログラム（タスクに割付）で使用可能（ST プログラムをタスクに割り当て可能）。 1 つのユーザプログラム内で、他の言語と任意に組み合わせることができます。 これにより、数値演算や文字列処理は ST プログラムで記述し、他の処理はラダー、SFC プログラムで記述することができます。 注：ST プログラムは、CS/CJ シリーズ ユニット Ver.4.0 以降のみ対応。 CP シリーズでは使用できません。

● ファンクションブロック定義、ST プログラムの照合が可能

従来 (Ver.7.0)	今回 (Ver.7.2)
・ファンクションブロック定義の照合はできませんでした。	・ファンクションブロック定義の照合が可能です。 これにより、ファンクションブロック定義間のプログラムの差異を簡単に確認できます。 ・ST プログラムの照合も可能です。

■ Ver.6.1 から Ver.7.0 への機能強化

[ラダープログラムをファンクションブロック化するための便利機能の提供]

従来 (Ver.6.1)	今回 (Ver.7.0)
ラダープログラムをファンクションブロック化する場合、ラダープログラムをコピーし、ファンクションブロック定義に貼り付けることで行っていました。その際、プログラム中での変数やアドレスの使用状況を見て、ファンクションブロックの変数テーブル上に、入力変数、内部変数、出力変数を仕分けしながら、手動で登録することが必要でした。	ラダープログラム中でファンクションブロック化した1つまたは複数の回路を選択し、[ファンクションブロック (ラダー) 生成] を選択することで、プログラム中での変数やアドレスの使用状況にあわせて、変数の用途を自動で割り当て (後で必要に応じて変更可能)、ファンクションブロック定義を生成することができます。これにより、既存ラダープログラムの資産を、簡単にファンクションブロック化することができます。

[FB オンラインエディット]

従来 (Ver.6.1)	今回 (Ver.7.0)
運転中の PLC のファンクションブロック定義 (アルゴリズムおよび変数テーブル) を変更することはできませんでした (インスタンスの入出力パラメータの変更は可能でした)。	運転中の PLC のファンクションブロック定義 (アルゴリズムおよび変数テーブル) を変更することができます*1。これにより、24 時間稼動など、運転を停止できない装置においても、ファンクションブロック定義自体のデバッグ/変更が可能となります。 操作: ワークスペース上のファンクションブロック定義を選択し、ポップアップメニューから [FB オンラインエディット] [開始] を選択することによります。 *1: インスタンスの追加は行えません。 注: 本機能は、CX-Simulator では使用不可。

[ファンクションブロック内の ST 言語で、STRING 型のデータ型、文字列処理関数をサポート]

従来 (Ver.6.1)	今回 (Ver.7.0)
- ST 言語内で、STRING 型 (文字列型) を使用することはできませんでした (*1)。 - ST 言語用文字列処理関数はありませんでした。 - ラダー言語で、表示用メッセージ処理や無手順通信処理などをするには、基本的に (*1) ユーザが ASCII コードのコードやサイズを意識して、文字列処理命令、データ変換命令、シリアル通信命令などを実行する必要がありました。 *1: ユーザが CX-Programmer の PLC メモリ機能を使用して、I/O メモリ上に文字列(テキスト)を入力することは可能。ただし、I/O メモリ上のデータサイズを意識することが必要。	- ファンクションブロックで使用する ST 言語内で、データ型として STRING 型 (文字列型) を使用することができます。これにより、例えば、ある変数に文字列を代入する式 (例: a := '@READ';) を記載するだけで、文字列 (ASCII コード) がいった変数を簡単にセットできます。この際、文字列の ASCII コードサイズを意識することは不要です。 - ST 言語用文字列処理関数をサポートしました (文字列抽出、結合、検索など)。これにより、文字列の加工処理、表示用メッセージ処理を、ファンクションブロック内の ST 言語を使って、簡単に実行できます。 - 文字列の送信/受信などの通信関数もサポートしました。これにより、無手順通信処理を、ファンクションブロック内の ST 言語を使って、簡単に (ASCII コードを意識せずに) 実行することができます。

[入出力変数をサポート]

従来 (Ver.6.1)	今回 (Ver.7.0)
- ファンクションブロック内で、入出力変数を使用することはできませんでした (入力変数、内部変数、出力変数のみ)。 - 入力変数は、配列指定ができません。 - 入力パラメータから入力変数へは、値が渡されます。	- ファンクションブロック内で、入出力変数を使用することができます。 - 入出力変数は、配列指定が可能です。 - 入力パラメータから入出力変数へは、値でなくアドレスが渡されます。 - そのため、ファンクションブロック内で配列指定の入出力変数を使用し、入力パラメータに、大量のデータを簡単にファンクションブロックに渡すことができます。

■ Ver.6.0 から Ver.6.1 への機能強化

[対応 PLC 機種]

- ・ NSJ シリーズに対応
NSJ シリーズに対応しました。
PLC 機種として「NSJ」、CPU 形式として「G5D」を選択します。
- ・ FQM1 シリーズ ユニット Ver.3.0 以降に対応
フレキシブルモーションコントローラ FQM1 の新機種(コーディネータモジュール形 FQM1-CM002、モーションコントロールモジュール形 FQM1-MMA22/MMP22)に対応しました。

[インスタンス内 ST/ラダープログラムのシミュレーション機能]

従来 (Ver.6.0)	今回 (Ver.6.1)
CX-Simulator を使用することによって、ラダープログラムのステップ実行、連続ステップ実行、スキャン実行、I/O ブレークポイント条件設定が可能でした。	CX-Programmer の機能として、ステップ実行、連続ステップ実行、スキャン実行、ブレークポイント設定が可能です。 いずれも、ラダープログラム、ファンクションブロック内ラダー/ST プログラムで可能です。 注 1：これらの機能を使用する場合は、別途 CX-Simulator Ver.1.6 以降がインストールされている必要があります。 注 2：I/O ブレーク条件設定はできません。

[ファンクションブロック機能の強化]

● ファンクションブロック内 ST プログラムのモニタが可能

従来 (Ver.6.0)	今回 (Ver.6.1)
オンラインでプログラムをモニタ中に、ファンクションブロックインスタンスの内部の ST プログラムのモニタをすることはできませんでした (ファンクションブロック定義のプログラム内容の確認およびファンクションブロック内ラダープログラムの導通モニタは可能)。	プログラムをモニタ中に、ファンクションブロックインスタンスの内部の ST プログラムをモニタすることができます。 ファンクションブロックインスタンスをダブルクリックするか、右クリックし [FB モニタ] を選択することで、可能です。 このとき、現在値変更、強制セット/リセットが可能です。 注：オンラインエディットはできません。

● ファンクションブロックのパスワードプロテクト機能

従来 (Ver.6.0)	今回 (Ver.6.1)
ファンクションブロックプロパティにて、ファンクションブロック定義内のプログラムを表示不可にすることのみ可能でした。	以下の 2 種類のパスワードプロテクトを設定することが可能です。 ・ 書込、表示とも禁止にするパスワードプロテクト ・ 書込のみ禁止にするパスワードプロテクト

■ Ver.5.0 から Ver.6.0 への機能強化

■ ファンクションブロック機能の強化

● ファンクションブロックのネスティングが可能

従来 (Ver.5.0)	今回 (Ver.6.0)
ファンクションブロックから、他のファンクションブロックを呼び出す（ネスティングする）ことができませんでした。	ファンクションブロックから他のファンクションブロックを呼び出すことが可能となりました。最大ネスティング数は8段（プログラムからファンクションブロックを呼び出すことを含む）です。呼び出し元ファンクションブロックと呼び出し先ファンクションブロック内の言語の組み合わせは、ラダー言語/ST 言語のいずれでも可能です。なお、ファンクションブロックのネスティングの階層関係をツリー表示することもできます。呼び出している（ネストしている）ファンクションブロック定義を含めて、1つのファンクションブロックライブラリファイル (.cxf) に保存されます。

● ファンクションブロック内ラダープログラムの導通モニタが可能

従来 (Ver.5.0)	今回 (Ver.6.0)
オンラインでプログラムをモニタ中に、ファンクションブロックインスタンスの内部のラダー図の導通状態をモニタすることはできませんでした（ファンクションブロック定義のプログラム内容の確認のみ可能）。	プログラムをモニタ中に、ファンクションブロックインスタンスの内部のラダー図を導通モニタすることができます。ファンクションブロックインスタンスをダブルクリックするか、右クリックし [FB ラダーモニタ] を選択することにより可能です。このとき、導通モニタ、現在値変更、強制セット/リセット、微分モニタが可能です。 注：オンラインエディット、タイマ/カウンタの設定値変更はできません。

● ファンクションブロックインスタンス内の変数をウォッチウィンドウ上に簡単に登録し、変数をモニタ可能

従来 (Ver.5.0)	今回 (Ver.6.0)
ファンクションブロックインスタンス内の変数をウォッチウィンドウに登録するには、ウォッチウィンドウを表示させ、ウォッチウィンドウをダブルクリックし、モニタしたい変数をプルダウンリストから選択することが必要でした。	ファンクションブロックインスタンス内の複数の変数を、ウォッチウィンドウに簡単に一括登録することが可能となりました。以下のいずれかの操作によって、[FB 変数登録ダイアログ] が表示されるので、そこから一括登録します。 ・ファンクションブロックインスタンスを右クリックし [ウォッチウィンドウへ登録] を選択 ・プログラム上または変数テーブル上のファンクションブロックインスタンスをウォッチウィンドウにコピー&ペーストまたはドラッグ&ドロップする。 ・ウォッチウィンドウの空き行にてポップアップメニューから [ウォッチウィンドウへ登録] を選択

● ファンクションブロック関連のその他強化機能

- ・ファンクションブロック内ラダープログラムで、クロスリファレンスポップアップ機能に対応しました。
- ・ST エディタ上のポップアップメニューから、ST 言語のヘルプが起動可能となりました。
- ・ファンクションブロックインスタンスをダブルクリックすることで、ファンクションブロック定義を開くようにしました。
- ・ファンクションブロックインスタンスのパラメータ入力確定後、カーソルが自動的に下に移動するように変更しました。

ファンクションブロック編	第1章 ファンクションブロックの概要	1
	第2章 ファンクションブロックの仕様	2
	第3章 ファンクションブロックの作成	3
ストラクチャードテキスト編	第4章 ST 言語の概要	4
	第5章 ST 言語の仕様	5
	第6章 ST プログラムの作成	6
付録		付

目次

はじめに	2
関連マニュアル	4
安全上のご注意	8
安全上の要点	13
マニュアル改訂履歴	15
バージョンアップのご案内	16
目次	24

第1章 ファンクション ブロックの概要 1-1

1-1 ファンクションブロック機能とは	1-2
1-1-1 CX-Programmer Ver.5.0 以降および CJ2 CPU ユニット、 CS/CJ シリーズ CPU ユニット ユニット Ver.3.0 以降、CP シリーズ、 NSJ、FQM1 のファンクションブロック機能の概要と特徴	1-2
1-1-2 ファンクションブロック関連の仕様	1-3
1-1-3 CX-Programmer Ver.6.0 以降で作成するファイル	1-5
1-1-4 CX-Programmer Ver.5.0 以降の ファンクションブロック機能に関連するメニュー	1-6
1-2 ファンクションブロックとは	1-9
1-2-1 ファンクションブロックの概要	1-9
1-2-2 ファンクションブロックによるメリット	1-10
1-2-3 ファンクションブロックの構成	1-11
1-3 変数とは	1-18
1-3-1 概要	1-18
1-3-2 変数の種類と属性	1-19
1-3-3 変数の属性	1-20
1-3-4 変数の属性と変数種類の関係	1-21
1-3-5 変数の使用アドレスの内部割当	1-22
1-4 ファンクションブロック定義のライブラリ化	1-24
1-5 使用手順の概要	1-25
1-5-1 ファンクションブロックの作成から実行まで	1-25
1-5-2 ファンクションブロックの再利用	1-26

第2章 ファンクションブロックの仕様 2-1

2-1 ファンクションブロックの仕様	2-2
2-1-1 ファンクションブロックの仕様	2-2
2-1-2 ファンクションブロックの構成要素	2-4
2-2 ファンクションブロックで使用するデータ型	2-15

2-2-1	基本データ型.....	2-15
2-2-2	派生データ型.....	2-15
2-3	インスタンスの仕様.....	2-16
2-3-1	インスタンスの構成要素.....	2-16
2-3-2	パラメータの仕様.....	2-22
2-3-3	動作仕様.....	2-23
2-4	制限事項.....	2-25
2-4-1	ラダー言語使用時の命令制限.....	2-25
2-4-2	ST 言語時の命令制限.....	2-28
2-4-3	ラダー/ST 言語の共通の制限.....	2-28
2-5	ファンクションブロック使用のガイドライン.....	2-32
2-5-1	変数のデータ型の決め方.....	2-32
2-5-2	変数種類（どれを入力変数、出力変数、入出力変数、外部変数、 内部変数とするか）の決め方.....	2-32
2-5-3	内部変数の AT 指定.....	2-35
2-5-4	入力変数または内部変数の配列型指定.....	2-36
2-5-5	高機能ユニットの割付アドレスの指定方法 （号機 No./ユニット番号から自動算出）.....	2-39
2-5-6	インデックスレジスタの使用方法.....	2-40
2-6	オペランドが複数ワードの先頭/最終アドレスを指定している命令の 注意事項.....	2-43
2-7	命令の使用可否とオペランドごとの使用変数の制限一覧.....	2-48
2-8	CPU ユニットのファンクションブロックの仕様.....	2-49
2-8-1	仕様.....	2-49
2-8-2	タイマ系命令の動作について.....	2-54
2-9	ファンクションブロック使用時のプログラムステップ数と インスタンス実行時間について.....	2-55
2-9-1	ファンクションブロック使用時の、プログラムステップ数について.....	2-55
2-9-2	ファンクションブロック使用時のインスタンス実行時間について.....	2-57
第 3 章 ファンクション ブロックの作成.....		3-1
3-1	使用手順フロー.....	3-2
3-2	操作方法.....	3-4
3-2-1	プロジェクトを作成する.....	3-4
3-2-2	ファンクションブロック定義を新規作成（挿入）する.....	3-4
3-2-3	ファンクションブロック定義を作成する（ユーザ作成の場合のみ）..	3-9
3-2-4	ファンクションブロック定義をプログラム上に貼り付ける （インスタンスを生成する）.....	3-22
3-2-5	ファンクションブロックを、外部と入出力割付する （ [Enter] キーでパラメータを設定する）.....	3-24
3-2-6	変数の内部割当範囲（FB インスタンスエリア）を設定する.....	3-27
3-2-7	各変数の内部割当アドレスを確認する.....	3-30
3-2-8	ファンクションブロック定義をコピーし、編集する.....	3-32
3-2-9	インスタンスから元のファンクションブロック定義を確認表示する ..	3-32

3-2-10	インスタンスの階層構造などの、インスタンスの情報を確認する	3-33
3-2-11	ファンクションブロックの使用量を確認する	3-34
3-2-12	ファンクションブロック定義をコンパイル（プログラムチェック）する	3-34
3-2-13	ファンクションブロック定義を印刷する	3-36
3-2-14	ファンクションブロック定義にパスワードプロテクトをかける	3-37
3-2-15	ファンクションブロックの照合	3-40
3-2-16	ファンクションブロック定義をファイルに保存して再利用する	3-40
3-2-17	プログラムを実機 PLC へダウンロード／アップロードする	3-41
3-2-18	ファンクションブロックをモニタ/デバッグする	3-42
3-2-19	ファンクションブロック定義のオンラインエディット	3-48

ストラクチャードテキスト編 第4章 ST 言語の概要 4-1

4-1	ST 言語とは	4-2
4-1-1	概要	4-2
4-2	CX-Programmer の仕様	4-3
4-2-1	ST プログラム（ST タスク）が使用可能な PLC 機種	4-3
4-2-2	仕様一覧	4-3

ストラクチャードテキスト編 第5章 ST 言語の仕様 5-1

5-1	ストラクチャードテキスト（ST）言語の仕様	5-2
5-1-1	ストラクチャードテキスト（ST）言語とは	5-2
5-2	ST プログラムで使用するデータ型	5-3
5-2-1	基本データ型	5-3
5-2-2	派生データ型	5-4
5-3	ST プログラムの入力について	5-5
5-3-1	記述上のルール	5-5
5-3-2	CX-Programmer の ST 入力画面での表示	5-8
5-4	ST 言語の構成	5-9
5-4-1	文の一覧	5-9
5-4-2	変数について	5-9
5-4-3	定数入力について	5-10
5-4-4	演算子一覧	5-10
5-4-5	標準関数一覧	5-11
5-4-6	オムロン拡張関数一覧	5-16
5-5	各構文の説明	5-19
5-5-1	代入文	5-19
5-5-2	制御文	5-20
5-6	ST 言語プログラム使用例	5-39
5-6-1	ファンクションブロックによる ST 言語プログラム使用例	5-39
5-7	制限事項	5-40
5-7-1	制限事項	5-40
5-7-2	Q&A（参考）	5-41

第6章 ST プログラムの作成.....	6-1
6-1 操作方法	6-2
6-1-1 プロジェクトを作成する	6-2
6-1-2 ST プログラムの新規作成（挿入）	6-2
6-1-3 ST プログラムのタスクへの割り付け	6-3
6-1-4 ST プログラムを作成する	6-5
6-1-5 ST プログラムをコンパイル（プログラムチェック）する	6-9
6-1-6 プログラムを実機 PLC へダウンロード/アップロードする	6-10
6-1-7 ST プログラムの照合	6-10
6-1-8 ST プログラムをモニタ/デバッグする	6-10
6-1-9 ST プログラムのオンラインエディット	6-12
付録.....	付-1
付-1 ファンクションブロックで使用するシステム定義の外部変数一覧.....	付-2
付-2 ST 言語のエラー一覧	付-3
付-3 関数の説明.....	付-5
付-3-1 標準関数.....	付-5
付-3-2 オムロン拡張関数.....	付-17

索引

ファンクションブロック編

第1章 ファンクション ブロックの概要

1-1 ファンクションブロック機能とは

1-1-1 CX-Programmer Ver.5.0 以降および CJ2 CPU ユニット、CS/CJ シリーズ CPU ユニット ユニット Ver.3.0 以降、CP シリーズ、NSJ、FQM1 のファンクションブロック機能の概要と特徴

1-1 ファンクションブロック機能とは

1-1-1 CX-Programmer Ver.5.0 以降および CJ2 CPU ユニット、CS/CJ シリーズ CPU ユニット ユニット Ver.3.0 以降、CP シリーズ、NSJ、FQM1 のファンクションブロック機能の概要と特徴

CX-Programmer Ver.5.0 以降は、IEC 61131-3 準拠のファンクションブロックの利用が可能な、プログラミングツールです。ファンクションブロック機能は、PLC 機種としては CS/CJ シリーズ CPU ユニット ユニット Ver.3.0 以降、CP シリーズ、NSJ、および FQM1 をサポートしています。

以下の特徴があります。

- ファンクションブロックを利用することで、ユーザ定義の処理をブロック化することができます。

- ファンクションブロックのアルゴリズムは、ラダー言語または ST (構造化テキスト) 言語^{*1}で記述できます。

- ・ラダー言語の場合、とくに既存の CX-Programmer で作成したラダープログラムを、コピー (カット) & ペーストで再利用することができます。
- ・ST 言語の場合、ラダー言語では記述しにくい数値演算処理を容易に記述できます。

^{*1}: ST (構造化テキスト) 言語とは、IEC 61131-3 で規定されている、工業制御 (おもにプログラマブルコントローラ) 用の高級言語です。CX-Programmer でサポートしている ST 言語は、IEC-61131-3 に準拠しています。

- 変数の宣言をテキストで記述する必要がない (変数テーブルへの登録方式の) ため、ファンクションブロックの作成が容易です。

ラダーで変数を記述するときにその都度、自動的に変数登録することができます。また、変数テーブルに登録後、ラダーまたは ST 上で、登録した変数を記述することもできます。

- 作成した 1 つのファンクションブロックを、1 つのファイルとしてライブラリ化できるため、定型的な処理の再利用が容易です。

- 作成した 1 つのファンクションブロックに対するプログラムチェックをかけることができ、1 つのファンクションブロック = ライブラリとしての信頼性を容易に確保できます。

- ファンクションブロック (ラダー言語または ST 言語) を含むプログラムも、ファンクションブロックを含まない通常のプログラムと同様、ダウンロード / アップロードできます。

ただし、ファンクションブロックを含むタスクを、タスク単位でダウンロードすることはできません (アップロードは可能です)。

- 配列型 (一次元配列) の変数をサポートしているため、アプリケーションに応じたデータハンドリングが容易となります。

参考: IEC61131 規格は、プログラマブルコントローラ (PLC) の国際標準規格として International Electro-technical Commission (IEC) が定めたものです。7 つの Part に分かれており、この中の Part3 プログラム言語 (IEC 61131-3) で、PLC のプログラミングについて規定しています。

- ファンクションブロック (ラダー言語または ST 言語) 内で他のファンクションブロック (ラダー言語または ST 言語) を呼び出すことができます (最大 8 段まで、ラダー / ST 言語の呼び出しの組み合わせは自由)

1-1-2 ファンクションブロック関連の仕様

以下に記載している仕様以外は、CX-Programmer のオペレーションマニュアル (SBCA-337) を参照してください。

項目	仕様
形式	形 CXONE-AL□□D-V4
セットアップ	CX-One のセットアップディスクにて、CX-Programmer を選択してください。
適用 PLC (プログラマブルコントローラ) 機種 ※CS/CJ シリーズ CPU ユニット Ver.4.0 以降でサポートしているファンクションブロック/ストラクチャードテキスト機能については、CS/CJ シリーズ CPU ユニット Ver.3.0 以前、CP シリーズ PLC、NSJ シリーズ PLC、FQM1 シリーズ PLC で使用することはできません。詳細については、「バージョンアップのご案内」(P16) を参照してください。	CS/CJ シリーズ CS1-H/CJ1-H CPU ユニット ユニット Ver.3.0 以降 PLC 機種：形式 CJ2H：形 CJ2H-CPU68/67/66/65/64/68-EIP/67-EIP/66-EIP/65-EIP/64-EIP CJ2M：形 CJ2M-CPU11/12/13/14/15/31/32/33/34/35 CS1G-H：形 CS1G-CPU45H/44H/43H/42H CS1H-H：形 CS1H-CPU67H/66H/65H/64H/63H CS1D-H：形 CS1D-CPU68HA/67HA CS1D-S：形 CS1D-CPU67SA/44SA CJ1G-H：形 CJ1G-CPU/45H/44H/43H/42H CJ1H-H：形 CJ1H-CPU67H-R/66H-R/65H-R/64H-R/67H/66H/65H CJ1M：形 CJ1M-CPU23/22/21/13/12/11 注：CX-Programmer Ver.5.0 以降で作成したファンクションブロックを含むユーザプログラムを、CS/CJ シリーズ CPU ユニット ユニット Ver.2.0 以前にダウンロードすると、すべてのインスタンスは、不正命令となり、ユーザプログラムは編集および実行することはできません。 CP1H：形 CP1H-XA/X/Y CP1L：形 CP1L-M/L CP1L-E：形 CP1L-EM/EL CP2E：形 CP2E-E/S CP2E-N：形 CP2E-N NSJ：G5D (形 NSJ5-TQ0□-G5D、形 NSJ5-SQ0□-G5D、形 NSJ8-TV0□-G5D、形 NSJ10-TV0□-G5D、形 NSJ12-TS0□-G5D で、共通) M3D (形 NSJ5-TQ0□-M3D、形 NSJ5-SQ0□-M3D、形 NSJ8-TV0□-M3D で、共通) FQM1-CM：形 FQM1-CM002 FQM1-MMA：形 FQM1-MMA22 FQM1-MMP：形 FQM1-MMP22 CS/CJ シリーズとしての機能制限事項： ファンクションブロック定義内での使用不可命令： ブロックプログラム命令 (BPRG/BEND)、サブルーチン命令 (SBS/GSBS/RET/MCRO/SBN)、ジャンプ命令 (JMP/CJP/CJPN)、ステップラダー命令 (STEP/SNXT)、都度リフレッシュ指定 (!) 命令、I/O リフレッシュ命令 (IORF)、1ms タイマ命令 (TMHH/TMHHX) (ただし、CJ1-H-R CPU ユニットの場合は使用可能) その他は、2-4 項「制限事項」を参照
ファンクションブロック定義作成	ファンクションブロック定義数 1 台の PLC (CPU ユニット) あたり、 CJ2H の場合： ・形 CJ2H-CPU6□(-EIP)：最大 2048 CJ2M の場合： ・形 CJ2M-CPU□1/□2/□3：最大 256 ・形 CJ2M-CPU□4/□5：最大 2048 CS1-H/CJ1-H の場合： ・CPU ユニット形式末尾 -CPU67H-R/66H-R/65H-R/64H-R/67H/66H/65H/45H/64H/44H：最大 1024 ・CPU ユニット形式末尾-CPU63H/43H/42H：最大 128 CS1D-H の場合： ・形 CS1D-CPU68HA/67HA：最大 1024 CS1D-S の場合： ・形 CS1D-CPU67SA/44SA：最大 1024 CJ1M の場合： ・形 CJ1M-CPU23/22/21/13/12/11：最大 128 CP1H の場合： ・形 CP1H-XA/X/Y：最大 128 CP1L の場合： ・形 CP1L-M/L：最大 128 CP1L-E の場合： ・形 CP1L-EM/EL：最大 128 CP2E の場合： ・形 CP2E-E/S：最大 64 CP2E-N の場合： ・形 CP2E-N：最大 64 NSJ の場合： ・形 NSJ□-□□□□-G5D：最大 1024、形 NSJ□-□□□□-M3D：最大 128 FQM1 の場合： ・形 FQM1-CM002、形 FQM1-MMA22、形 FQM1-MMP22：最大 128

1-1 ファンクションブロック機能とは

1-1-2 ファンクションブロック関連の仕様

1

ファンクションブロックの概要

項目	仕様	
ファンクションブロック定義作成	ファンクションブロック定義名	半角または全角文字。最大全角 32 文字。
	変数	変数名
		変数の種類
		ファンクションブロック定義で使用する変数（内部変数、外部変数、EN、ENO を除く）の数
		変数の使用アドレスの割当
		実アドレス（AT）指定
		配列指定
	言語	ラダーまたは ST（構造化テキスト）*1 で作成可能
インスタンス生成	インスタンス数	1 台の PLC（CPU ユニット）あたり、 CJ2H の場合： ・形 CJ2H-CPU6□(-EIP)：最大 2048 CJ2M の場合： ・形 CJ2M-CPU□1/□2/□3：最大 256 ・形 CJ2M-CPU□4/□5：最大 2048 CS1-H/CJ1-H の場合： ・CPU ユニット形式末尾 -CPU67H-R/66H-R/65H-R/64H-R/67H/66H/65H/45H/64H/44H：最大 2048 ・CPU ユニット形式末尾-CPU63H/43H/42H：最大 256 CS1D-H の場合 ・形 CS1D-CPU□□HA：最大 2048 CS1D-S の場合 ・形 CS1D-CPU□□SA：最大 2048 CJ1M の場合： ・形 CJ1M-CPU23/22/21/13/12/11：最大 256 CP1H の場合： ・形 CP1H-XA/X/Y：最大 256 CP1L の場合： ・形 CP1L-M/L：最大 256 CP1L-E の場合： ・形 CP1L-EM/EL：最大 256 CP2E の場合： ・形 CP2E-E/S：最大 128 CP2E-N の場合： ・形 CP2E-N：最大 128 NSJ の場合： ・形 NSJ□-□□□□-G5D：最大 2048、形 NSJ□-□□□□-M3D：最大 256 FQM1 の場合： ・形 FQM1-CM002、形 FQM1-MMA22、形 FQM1-MMP22：最大 256
	インスタンス名	半角または全角文字。最大全角 15000 文字。
ファイル保存機能	プロジェクトファイル	プロジェクトファイル（.exp/cxt）に、ファンクションブロック定義およびインスタンスが、含まれる
	プログラムファイル	ファイルメモリ用プログラムファイル（*.obj）に、ファンクションブロック定義およびインスタンスが、含まれる
	ファンクションブロック・ライブラリファイル	1 ファンクションブロック定義=1 ファイル（.cxf）で保存。他プロジェクトで再利用可能

*1：IEC 61131-3 準拠。ただし、代入文、選択文（CASE 文、IF 文）、繰り返し文（FOR 文、WHILE 文、REPEAT 文、EXIT 文）、RETURN 文、四則演算、論理演算、比較演算、数値演算関数、算術関数、標準文字列関数、数値文字列変更関数、オムロン拡張関数、コメントをサポートしています。詳細は、ストラクチャードテキスト編 第 5 章「ST 言語の仕様」を参照してください。

1-1-3 CX-Programmer Ver.6.0 以降で作成するファイル

1

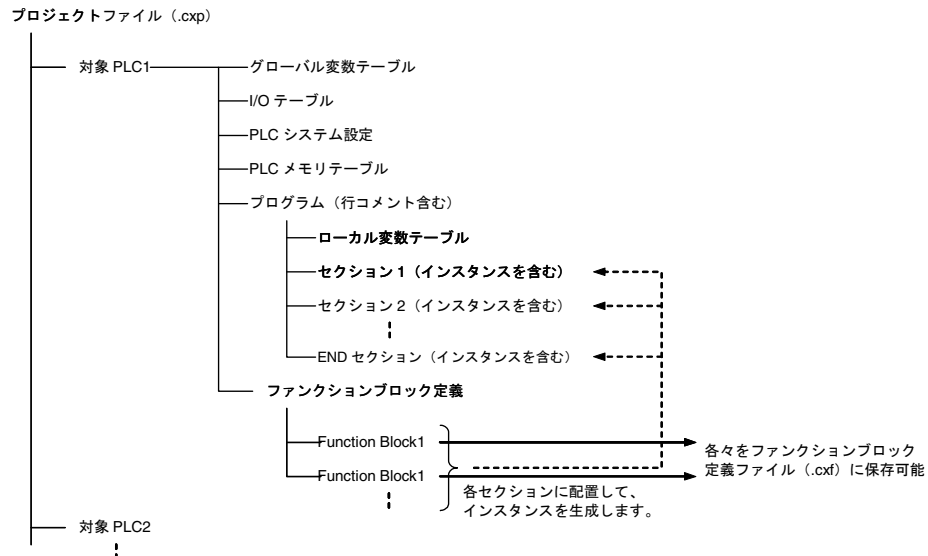
ファンクションブロックの概要

- ファンクションブロックを含むプロジェクトファイル (*.cpx) / ファイルメモリ用プログラムファイル (*.obj)

ファンクションブロック定義およびインスタンスを含むプロジェクトも、通常と同じプロジェクトファイル（拡張子.cxp のファイル）およびファイルメモリ用プログラムファイル (*.obj) として保存されます。

プロジェクトの内容は、以下のとおりです。

対象 PLC の下のプログラムと同等位置に、ファンクションブロック定義を作成します。



- ファンクションブロック・ライブラリファイル (*.cxf)

CX-Programmer Ver.6.0 以降では、プロジェクト上に作成したファンクションブロック定義を 1 定義=1 ファイルとして、保存して、他のプログラム上に読み出して再利用することができます。

注：ファンクションブロック内でネスティングしている場合、ネスティング（呼び出し）先の全てのファンクションブロック定義がこのファンクションブロック・ライブラリファイル (.cxf) に含まれます。

- ファンクションブロックを含むプロジェクトのテキストファイル (*.cxt)

CX-Programmer Ver.6.0 以降のプロジェクトファイル (.cpx) と同等の情報を、テキストファイル形式で CXT ファイル（拡張子.cxt）として保存することもできます。

1-1 ファンクションブロック機能とは

1-1-4 CX-Programmer Ver.5.0 以降のファンクションブロック機能に関連するメニュー

1

1-1-4 CX-Programmer Ver.5.0 以降のファンクションブロック機能に関連するメニュー

以下に、CX-Programmer Ver.5.0 以降のメニューの中で、ファンクションブロック機能に関連するメニューを示します。メニュー全体については、CX-Programmer オペレーションマニュアル（SBCA-337）を参照してください。

■ メインメニュー

メインメニュー	サブメニュー		ショートカット	機能
ファイル	ファンクションブロック	ファンクションブロックをファイルからロード		保存したファンクションブロック・ライブラリファイル（拡張子.cxf）を読み出します。
		ファンクションブロックをファイルへ保存		作成したファンクションブロック定義をファイル（「ファンクションブロック・ライブラリファイル」拡張子.cxf）に保存します。
編集	ファンクションブロック呼び出しの更新			インスタンス生成後、ファンクションブロック定義の入力変数、出力変数、または入出力変数を変更した場合（このときインスタンス左母線にエラー赤表示あり）、その変更内容をインスタンスに対して更新します（エラー赤表示を解除します）。
	下の階層へ			選択中のインスタンスに対応するファンクションブロック定義にジャンプします。
	ファンクションブロック（ラダー）生成			選択された回路をもとに、アドレスの使用状況を自動的に判断し、ファンクションブロック定義（ラダー）を生成します。
表示	FB モニタ			オンラインでプログラムのモニタ中に、インスタンス内ラダー図の導通モニタおよびSTの変数モニタをします。（Ver.6.1以降）
	下の階層へ			選択中のインスタンスのファンクションブロック定義の内容を（右側のビューに）表示する。（Ver.6.0以降）
	上の階層へ			ネスティング元のインスタンスのラダー図、またはSTに戻ります。（Ver.6.0以降）
	ウィンドウ	FB インスタンスビューア		FB インスタンスビューアを表示します。 （ネスティング時のインスタンスの階層関係、インスタンス内変数の割当アドレスなどを表示）
挿入	ファンクションブロック呼び出し		F	プログラム（セクション）の現在のカーソル位置上に、インスタンスを配置します。
	ファンクションブロック パラメータ		P	入力変数の左隣の位置、または出力変数の右隣の位置にカーソルがある場合、入力変数または出力変数に対するパラメータを設定します。
PLC	メモリ割付	ファンクションブロック/SFC メモリ	ファンクションブロック/SFC アドレス割付	選択中のインスタンスの変数の内部割当範囲（FB インスタンスエリア）を設定します。
		ファンクションブロック/SFC メモリ	ファンクションブロック/SFC メモリ情報	選択中のインスタンスの変数の、内部割当アドレスの使用状況を確認します。
			ファンクションブロックインスタンスアドレス	選択中のインスタンスの各変数の、内部割当アドレスを確認します。
			ファンクションブロック/SFC メモリ割付の最適化	変数の内部割当アドレスの割り当てを最適化します。
プログラム	オンラインエディット	開始		FB オンラインエディットを開始します。
		変更を転送		FB オンラインエディットでの変更箇所を転送します。
		取消		FB オンラインエディットでの編集内容をキャンセルします。
		FB ソース転送		FB ソースのみ転送します
		FB/SFC/ST オンラインエディットアクセス権強制解放		他のユーザが獲得している FB/SFC/ST オンラインエディットのアクセス権を強制的に解放します。

1-1 ファンクションブロック機能とは

1-1-4 CX-Programmer Ver.5.0 以降のファンクションブロック機能に関連するメニュー

1

ファンクションブロックの概要

メインメニュー	サブメニュー	ショートカット	機能
シミュレーション	デバッグ	ブレークポイント 設定/解除	ブレークポイントの設定/解除を行う
		ブレークポイント 全解除	すべてのブレークポイントの解除を行う
		モード 実行 (モニタモード)	スキャンを連続して行う (ラダー実行エンジンの運転モードを「モニタ」モードにする)
		モード 停止 (プログラムモード)	シミュレータの動作モードをプログラムモードにする
		モード 一時停止	シミュレータの動作を一時停止にする
		ステップ実行 ステップ実行	シミュレータのプログラムを1ステップだけ実行する
		ステップ実行 ステップイン実行	ファンクションブロック呼出し命令の場合に、内部のプログラムのステップ実行に入る
		ステップ実行 ステップアウト実行	ファンクションブロックの内部プログラムのステップ実行中の場合に、ひとつ上の階層 (呼び出し元) に戻って一時停止する
		ステップ実行 連続ステップ実行	ステップ実行を一定の時間間隔で連続して行う
		ステップ実行 スキャン実行	1サイクル実行して一時停止する
		現在の実行位置を常に表示	ステップ実行または連続ステップ実行に合わせて、自動スクロールして、常に一時停止位置を表示する
		ブレークポイント一覧	設定されているブレークポイントの一覧を表示する (指定位置へのジャンプ操作が可能)
ツール	入力モード	スマートインプットモード	入力モードをスマートインプットモードに切り替える
		クラシックモード	入力モードをクラシックモードに切り替える

■ おもなポップアップメニュー (右クリックで表示されるメニュー)

● ファンクションブロック定義のポップアップメニュー

ポップアップメニュー		機能
ファンクションブロック挿入	ラダー	ラダーをアルゴリズムとしたファンクションブロック定義を作成（挿入）します。
	ストラクチャードテキスト	STをアルゴリズムとしたファンクションブロック定義を作成（挿入）します。
	ライブラリファイルからロード	ファンクションブロック定義を、ファンクションブロックライブラリファイル（*.cxf）から読み出します。

● 挿入した個々のファンクションブロック定義のポップアップメニュー

ポップアップメニュー		機能
開く		選択中のファンクションブロック定義の内容を右側ビューに表示します。
ファンクションブロックをファイルへ保存		選択中のファンクションブロック定義をファイルに保存します。
コンパイル（プログラムチェック）		選択中のファンクションブロック定義をコンパイルします。
FB オンライン エディット	開始	FB オンラインエディットを開始します。
	変更を転送	FB オンラインエディットでの変更箇所を転送します。
	取消し	FB オンラインエディットでの編集内容をキャンセルします。
	FB ソース転送	FB ソースのみ転送します
	FB オンラインエディットアクセス権強制開放	他のユーザが獲得している FB オンラインエディットのアクセス権を強制的に開放します。

● ファンクションブロック変数テーブルのポップアップメニュー

ポップアップメニュー		機能
編集		変数を編集します。
変数追加		変数を最下行に追加します。
変数挿入	上側	変数を、現在カーソルがある位置の上に挿入します。
	下側	変数を、現在カーソルがある位置の下に挿入します。
切り取り		変数を切り取ります。
コピー		変数をコピーします。
貼り付け		変数を貼り付けます。
検索		変数を検索します。対象：変数名、変数コメント、すべて（文字列）
置換		変数を置換します。

1-1 ファンクションブロック機能とは

1-1-4 CX-Programmer Ver.5.0 以降のファンクションブロック機能に関連するメニュー

1

削除	変数をコピーします。
名前の変更	変数の名称のみを変更します。

● インスタンスのポップアップメニュー

ポップアップメニュー	機能
編集	インスタンス名を変更します。
ファンクションブロック呼び出しの更新	インスタンス生成後、ファンクションブロック定義の入力変数、出力変数、または入出力変数を変更した場合（このときインスタンス左母線にエラー赤表示あり）、その変更内容をインスタンスに対して更新します（エラー赤表示を解除します）。
FB ラダーモニタ	オンラインでプログラムのモニタ中に、インスタンス内のラダー図の導通モニタをします（Ver.6.0 以降）。
FB モニタ	オンラインでプログラムのモニタ中に、インスタンス内ラダー図の導通モニタおよび ST の変数モニタをします。（Ver.6.1 以降）
下の階層へ	選択中のインスタンスのファンクションブロック定義の内容を（右側のビューに）表示する。（Ver.6.0 以降）
上の階層へ	ネスティング元のインスタンスのラダー図、または ST に戻ります。（Ver.6.0 以降）
ウォッチウィンドウへ登録	選択しているインスタンス内の変数をウォッチウィンドウへ登録するための「FB 変数登録」ダイアログボックスを表示します。
ファンクションブロック定義	選択中のインスタンスの、元のファンクションブロック定義の内容を（右側ビューに）表示します。

■ ショートカットキー

- [F] キー：ファンクションブロック定義をプログラム上に貼り付けるとき
ラダーセクションウィンドウ上で、ファンクションブロックのインスタンス（コピー）を作成したい位置にカーソルを置いて、[F] キーを押します。[挿入]メニュー | [ファンクションブロック呼び出し] と同じです。
- [Enter] キー：パラメータを入力するとき
入力変数または入出力変数の左隣の位置、または出力変数の右隣の位置にカーソルを置いて、[Enter] キーを押します。[挿入]メニュー | [ファンクションブロック パラメータ] と同じです。

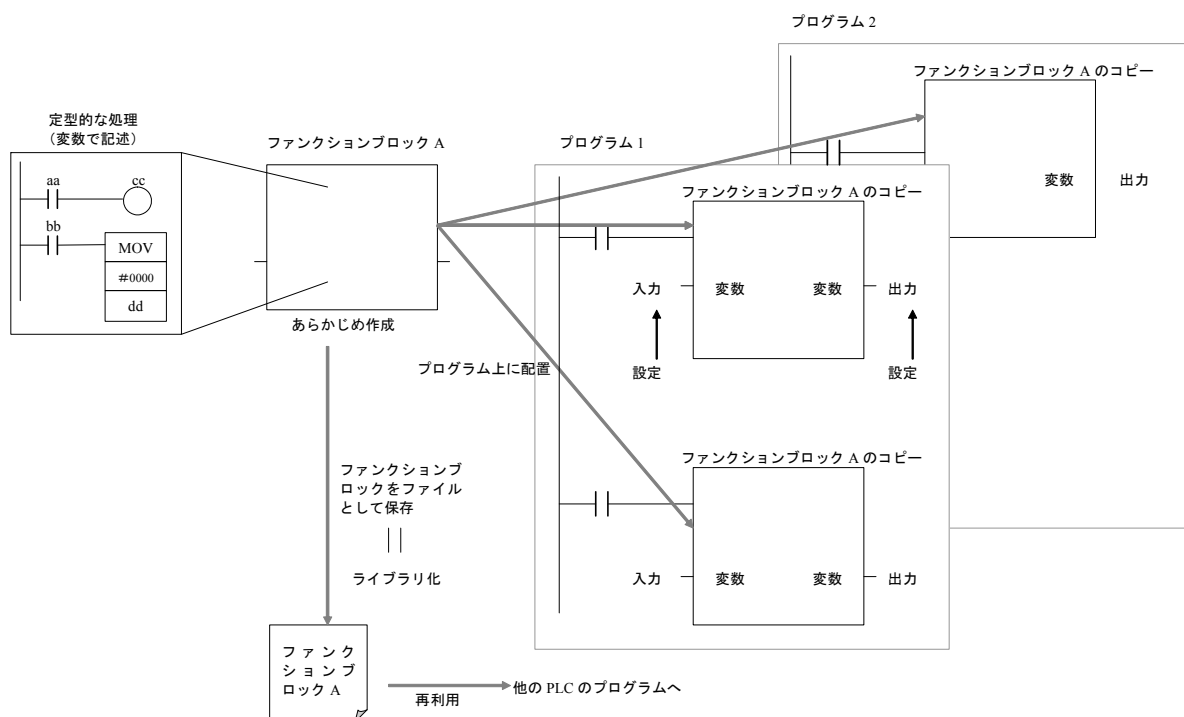
1-2 ファンクションブロックとは

1-2-1 ファンクションブロックの概要

「ファンクションブロック」とは、ある定型的な処理（機能）を1つのかたまり（ブロック）としてあらかじめ作成しておき、ユーザは、それをプログラムに配置して、入出力を設定するだけで使用することができる、プログラム要素です。

定型的な処理は、すべて基本的に実アドレスでなく、「変数」で記述しておきます。ユーザは、その「変数」にパラメータ（アドレスまたは値）を設定して使用します。その「変数」自体の使用アドレスは、プログラムに配置するごとに、システム（CX-Programmer）によって、自動的に割り当てられます。

とくに、CX-Programmerでは、1つのファンクションブロックを、1つのファイルとして保存し、他のPLC用プログラムで再利用することができるため、定型的な処理（機能）のライブラリ化が可能です。



1-2-2 ファンクションブロックによるメリット

標準的な複数回路をファンクションブロックとして作成/ファイル化しておけば、それを配置して、入出力にパラメータを設定するだけで、簡単に、複雑な回路を再利用することができます。

これにより、プログラムの作成およびデバッグ工数を削減し、コーディングミスが少なくかつ第三者にもわかりやすい、プログラムを作成することが可能です。

■ 構造化プログラミング

ファンクションブロックによるプログラムの構造化によって、ユーザの設計品質を上げ、開発工数を低減できます。

● ブラックボックス化により、視認性を向上

プログラム上では、入出力部だけが変数名で表示されるため、プログラムを記述するときまたは読むときに、適切なブラックボックス化がなされ、余計な理解の工数がかかりません。

● 1つのファンクションブロックから複数の異なる処理を、簡単に作成

定型的な処理の中のパラメータ要素（たとえばタイマ時間、制御定数、速度設定、移動距離など）を入力変数としておくことで、1つのファンクションブロックから複数の異なる処理を、簡単に作成することができます。

● コーディングミスが減少

再利用可能な単位がデバッグ済みでかつ明確なため、コーディングミスを減らすことができます。

● ノウハウの隠ぺい化

ファンクションブロックに対して読出プロテクトを行うことにより、プログラムのノウハウ流出を防ぐことができます。

● データ保護が可能

ファンクションブロック内の変数に対して外部から直接アクセスすることができないため、データを保護することができます（不用意にデータが変更されることがありません）。

● 変数プログラミングで再利用性を向上

変数で記述しておくため、再利用するときに、（回路をコピーするときと異なり）アドレスの振り直しをする必要がありません。

■ ライブラリの作成

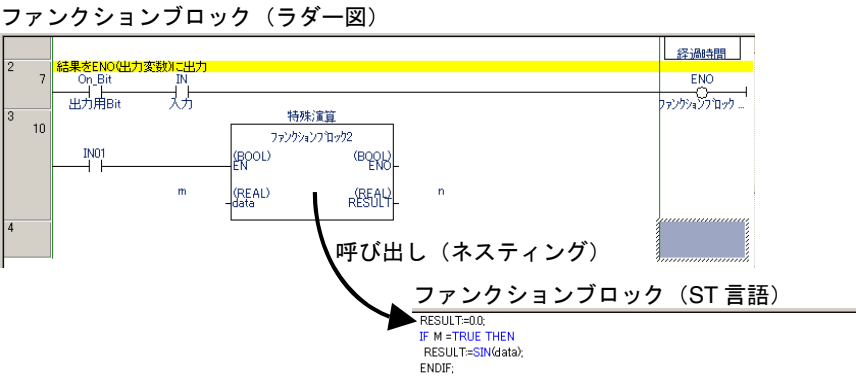
工程ごと、機械・装置ごと、制御部ごとなどの、独立性・再利用性が高い処理を、ファンクションブロック定義としてファイルに保存してライブラリ化できます。

とくに、実アドレスに依存しない変数名で作成しておけるため、必要時に、ファイルから読み出してプログラム上に配置するだけで、容易にプログラムを開発できます。

■ マルチ言語対応かつネスティング

ST 言語による数式表現ができます。

CX-Programmer Ver.6.0 以降では、ファンクションブロックのネスティングが可能です。これにより、たとえば、ラダー図のファンクションブロック内で、特殊な演算処理のみを ST 言語で表現することができます。

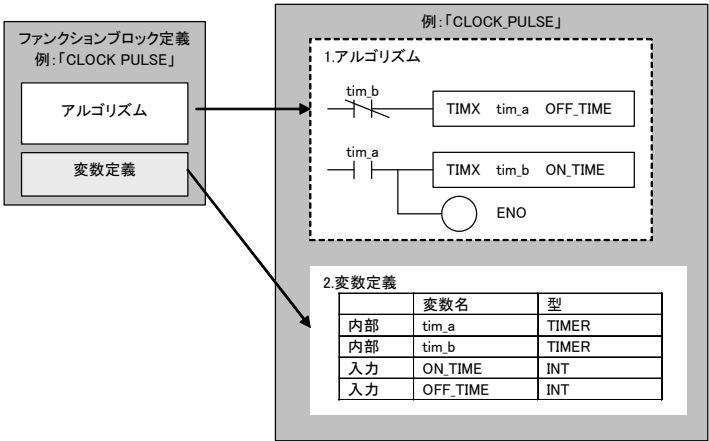


1-2-3 ファンクションブロックの構成

ファンクションブロックは、あらかじめ作成した「ファンクションブロック定義」と、それを実際にプログラム上に配置した「インスタンス」から成ります。

■ ファンクションブロック定義とは

ファンクションブロック内に記述するプログラムを、「ファンクションブロック定義」と呼びます。
「ファンクションブロック定義」には、以下のように、アルゴリズムと変数定義を記載します。



- 1 「アルゴリズム」：
I/O メモリの実アドレスでなく、変数名で記述した定型的な回路。
CX-Programmer では、ラダーまたは ST（構造化テキスト）で記述できます。
- 2 「変数定義」：
変数の種類（入力用、出力用、入出力用、内部用）と属性（データ型など）を記述したテーブル。変数の詳細は、後述「変数とは」を参照。

● ファンクションブロック定義の数

ファンクションブロック定義数は、1 台の PLC（CPU ユニット）あたり、最大 64 から 2048 です（CPU ユニット形式によって異なります）。

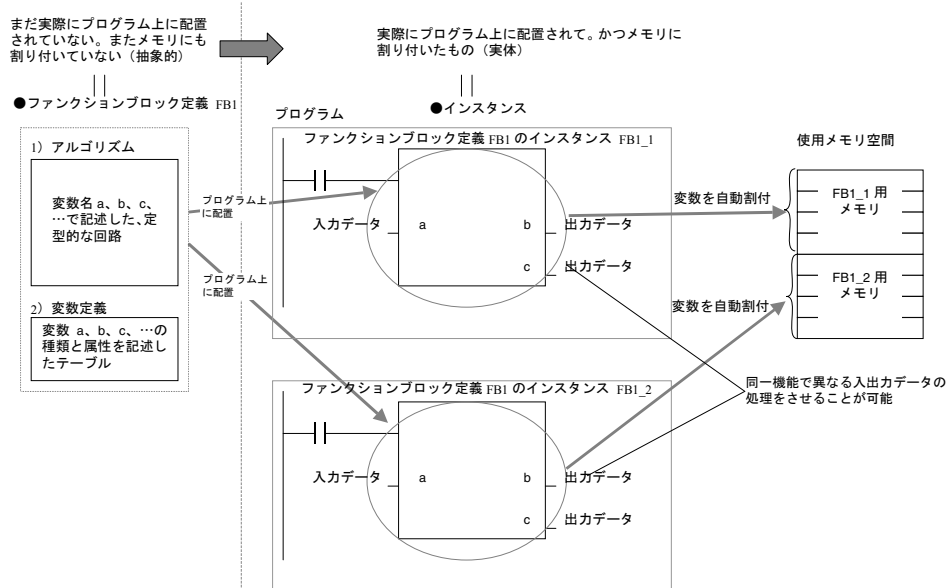
■ インスタンスとは

実際にファンクションブロックをプログラム内で使用するためには、ファンクションブロック定義のコピーを生成し、プログラム上に配置する必要があります。

ファンクションブロック定義をプログラム上に配置したものを、「インスタンス」と呼びます。

各インスタンスは、インスタンス名という識別子で区別します。

この「インスタンス」を生成することにより、1つのファンクションブロック定義を使用して、同一機能で異なる入出力データの処理をさせることができます。



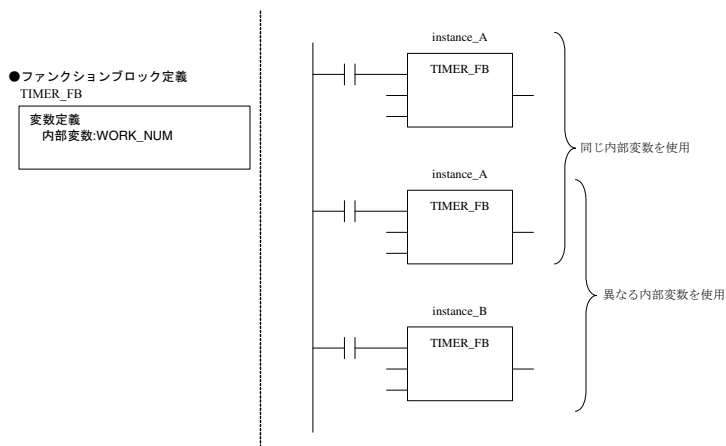
注：インスタンスは名前で管理します。同じ名前のインスタンスを複数箇所に配置することも可能です。

その場合、それぞれのインスタンスは同じ内部変数を使用します。

異なる名前のインスタンスは別の内部変数を持ちます。

例えば、タイマを内部変数として使用する FB を複数箇所で利用する場合は、異なるインスタンス名にする必要があります。同一のインスタンスを用いると、複数箇所で同じタイマを使用することとなり、タイマ二重使用となります。

内部変数を使用しない場合や、内部変数を一時的に使用し次にそのインスタンスを実行するときは初期化される場合は、インスタンス名を同じとすることでメモリの節約ができます。



● インスタンスの数

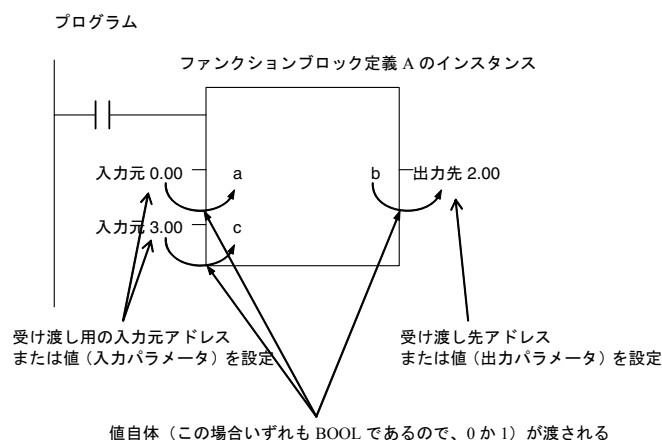
1つのファンクションブロック定義から複数のインスタンスを作成することができます。

作成可能なインスタンスの数は、1台のPLC（CPUユニット）あたり最大128から2048です（CPUユニット形式によって異なります。元としたファンクションブロック定義数、および配置先のタスク数とは無関係）。

● パラメータ

インスタンスを作成するごとに、インスタンスに入力データの値を渡したり、インスタンスからの出力データから値を取得するために、入力変数/出力変数/入出力変数との受け渡し用のI/Oメモリの実アドレス（または定数）を設定する必要があります。

この受け渡し用のアドレス（または定数）を「パラメータ」と呼びます。



入力変数/出力変数の場合

入力変数/出力変数の場合、受け渡し用の入力元アドレス（入力パラメータ）から、ファンクションブロックに対して渡されるデータは、そのアドレス内の（変数のデータ型で指定されたサイズの）値（データ）自体であって、アドレスではありません。同様に、ファンクションブロックから受け渡し用の出力先アドレス（出力パラメータ）に渡されるデータは、（変数のデータ型で指定されたサイズの）値（データ）自体であって、アドレスではありません。

1-2 ファンクションブロックとは

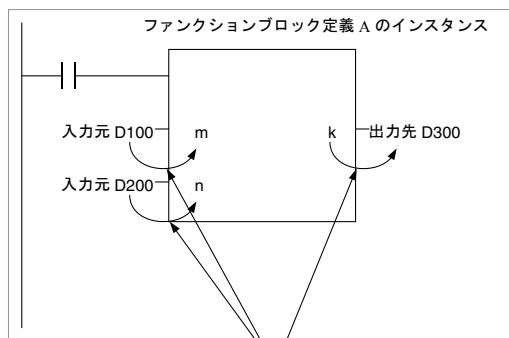
1-2-3 ファンクションブロックの構成

1

ファンクションブロックの概要

受け渡し用の入力元アドレス（入力パラメータ）、出力先アドレス（出力パラメータ）がCHアドレスの場合でも、以下のように、そのアドレスを先頭とした（変数のデータ型で指定されたサイズ分の）データ自体が渡されます。

プログラム



例)

m が WORD 型である場合は、D100 内の 1 ワードデータが変数に渡される。

n が DWORD 型である場合は、D200～D201 内の計 2 ワードデータが変数に渡される。

k が LWORD 型である場合は、変数の計 4 ワードデータが D300～D303 に渡される。

注 1：パラメータとして使用できる I/O メモリアドレスは、CIO（チャンネル I/O）、A（特殊補助リレー）、D（データメモリ）、E0～C（拡張データメモリ）、H（保持リレー）、W（内部補助リレー）のみです。

以下の指定はできません。

- ・インデックスレジスタ／データレジスタ（間接指定、直接指定とも）

- ・DM/EM 間接指定（@：BIN モード、*：BCD モードとも）

注 2：パラメータとして、プログラム側のローカル変数またはグローバル変数を指定することも可能です。ただし、受け渡し先または受け渡し元のファンクションブロック側の変数とデータのサイズが一致していることが必要です。

注 3：パラメータから入力変数に値（データ）を受け渡すタイミングは、インスタンス実行時のアルゴリズム処理前です。出力変数からパラメータに、値（データ）を受け渡すタイミングは、インスタンス実行時のアルゴリズム処理後です。したがって、アルゴリズムの中で、パラメータから入力変数へ、値を読み出すことはできません。

お願い

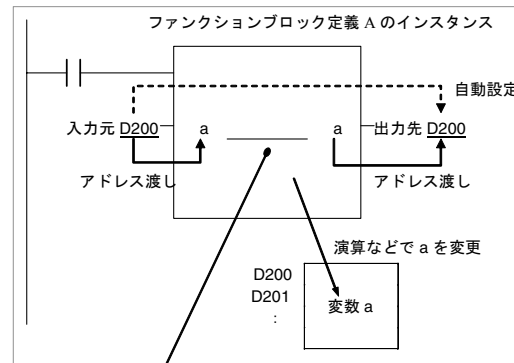
- ・入力パラメータに、アドレスを指定しても、入力変数には、そのアドレス内の値を渡します。アドレス情報自体を渡すことはできませんので、ご注意ください。
- ・アルゴリズムの中で（その実行サイクル内で）値を読出したい、または書き込みたい場合、パラメータを使用することはできません。その場合、変数を、内部変数として、後述の AT 指定（実アドレスへの割り当て）を使用してください。または、外部変数としてグローバル変数を参照してください。

入出力変数の場合

入出力変数の場合、入力パラメータには、アドレスを設定します（定数を設定することはできません）。

ファンクションブロックに対しては、その入力パラメータに設定したアドレス自体が渡されます。ファンクションブロック内で、この入出力変数を使用して演算などを行っている場合、その演算結果が、そのアドレス以降の変数サイズ分の I/O メモリに反映されます。

プログラム



ファンクションブロック内で入出力変数 a を使用して、演算を実行。アドレス D200 以降の変数 a のサイズ分の I/O メモリの値が変更される。

1-2 ファンクションブロックとは

1-2-3 ファンクションブロックの構成

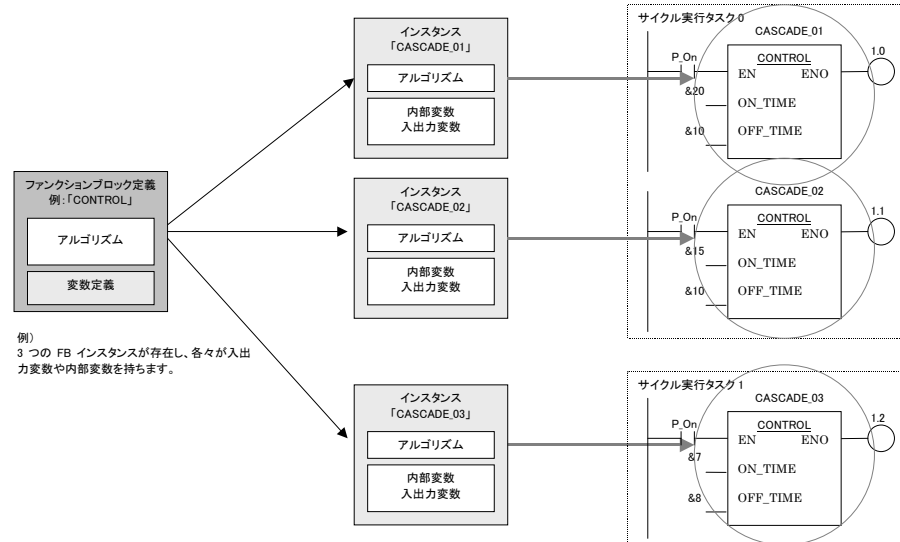
1

ファンクションブロックの概要

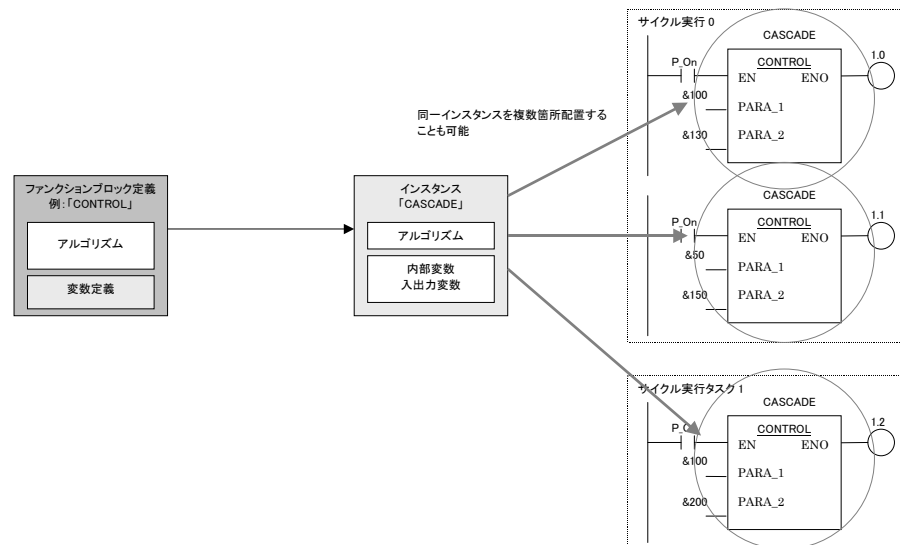
参考

- ・パラメータ的な要素（固定値など）をファンクションブロックの入力変数とし、インスタンスごとに入力変数の受け渡し値を変更することで、1つのファンクションブロックをもとにさまざまな処理を簡単に作成することができます。

例) 1つのファンクションブロック定義から、3つのインスタンスを作成した場合



- ・内部変数を使用しない場合や、内部変数を他の箇所でも使用しても影響がない場合は、同一のインスタンスを、プログラムの複数箇所でも使用することもできます。

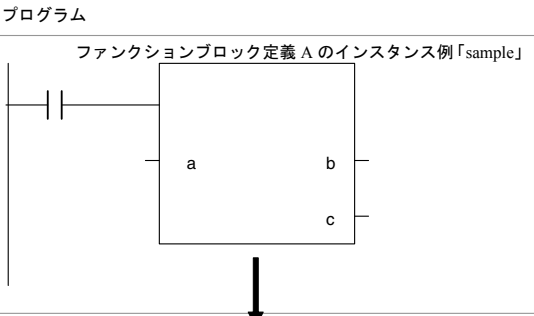


ただし、その場合、使用メモリが同一となる点に注意が必要です。

例) もしタイマ命令を含む同一インスタンスをプログラムの複数箇所で使用すると、同じタイマ番号を使用すること（「コイル二重使用」）になり、タイマ命令が同時に動作した場合、誤動作してしまいます。

● インスタンスの登録先

インスタンスは、インスタンス名を名称としてグローバル変数テーブルに登録されます。



グローバル変数テーブルに、インスタンス名を名称として、
インスタンスが登録されます。

グローバル変数テーブル

名称	データ型	アドレス／ 値	
sample	FB [FunctionBlock1]	N/A [Auto]	

↑ ↑
インスタンス名 タイプ欄に、FB の後、[] で使用する
 ファンクションブロック定義名が登録されます。

1-3 変数とは

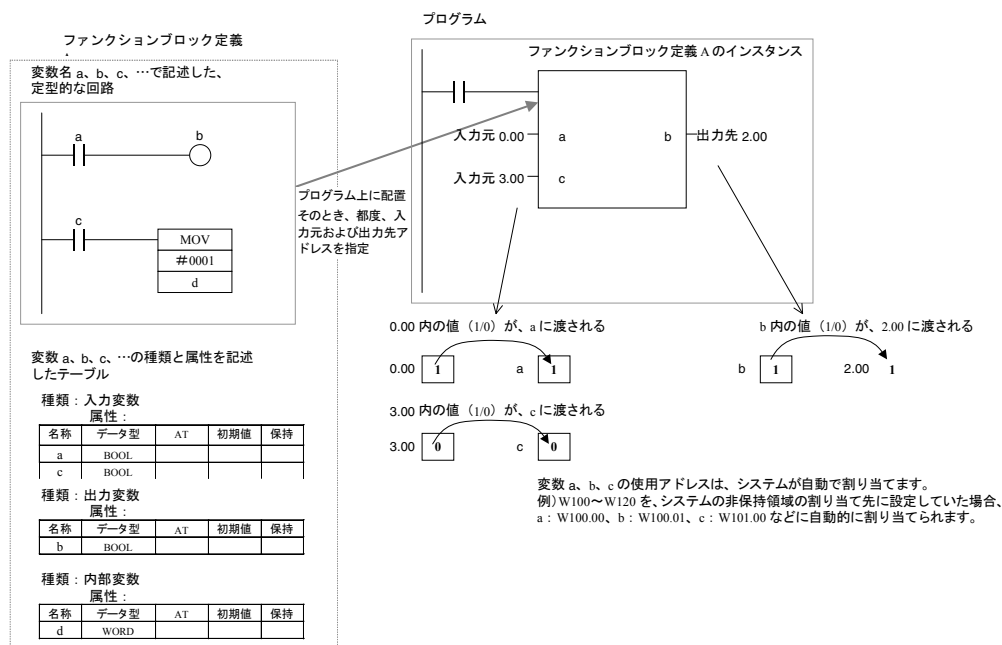
1-3-1 概要

ファンクションブロック内では、アドレス（注）を、I/O メモリの実アドレスでは記述せず、すべて変数名で記述します。

変数の使用アドレス自体は、インスタンスを作成するごとに、指定領域の I/O メモリの実アドレスが、システム（CX-Programmer）によって自動的に割り当てられます。したがって、ファンクションブロック内では、ユーザは、パソコン内でのメモリ割付と同様に、I/O メモリの実アドレスを意識する必要がありません。

ファンクションブロックは、この点（変数名で記述し、アドレスがブラックボックスである点）が、サブルーチンと大きく異なります。

例)



注：

- ・定数は、変数としては登録しません。命令のオペランドに直接入力します。
 - －ラダー言語の場合：10 進数は、&の後に数値を入力します。16 進数は、#の後に数値を入力します。
 - －ST 言語の場合：10 進数は、そのまま数値を入力します。16 進数は、16#の後に数値を入力します。
- ・例外的に、インデックスレジスタ（IR0～15）の直接/間接指定のアドレスおよびデータレジスタ（DR0～15）は、命令のオペランドに直接入力します。

1-3-2 変数の種類と属性

■ 変数の種類

変数には、以下の種類 (Usage) があります。

- | | |
|--------------------|--|
| 内部変数 (Internals) : | インスタンス内部でのみ使用する変数。
インスタンスの外部のパラメータからは、データ入力およびデータ出力ができない。 |
| 入力変数 (Inputs) : | インスタンスの外部のパラメータから、データを入力することができる変数。
デフォルトでは、入力条件データを入力するための"EN" (Enable) が生成されます。 |
| 出力変数 (Outputs) : | インスタンスの外部のパラメータへ、データを出力することができる変数。
デフォルトでは、インスタンスの実行状態を出力するための"ENO" (Enable Out) が生成されます。 |
| 入出力変数 (In Out) | インスタンスの外部のパラメータからデータを入力し、インスタンス内部での変更をパラメータに反映させることができる変数。
パラメータのアドレスが変数に渡されます。 |
| 外部変数 (Externals) : | ①コンディションフラグ、一部の特殊補助リレーなど、
CX-Programmer にあらかじめ登録されているシステム定義変数、および
②グローバル変数でユーザ定義されている変数を、インスタンス内部で使用するための変数。 |

各変数の種類 (Usage) ごとの説明は、2-1-2 項「ファンクションブロックの構成要素」の「■変数定義」の「●変数の種類」を参照してください。

1-3-3 変数の属性

変数の属性には、以下の内容があります。

■ 変数名 (Name)

ファンクションブロック内で変数を識別するための名称。

他のファンクションブロックで同一名を使用してもかまいません。

注：半角英数字・記号、全角文字・記号を使用可能。最大全角で 15000 文字。

ただし、数字からはじめることは不可。また、半角のアンダーバー (_) を 2 個繰り返すことは不可。 ,IRO~15 などのインデックスレジスタの文字列と同一も不可。その他の禁則は、2-1-2 項「ファンクションブロックの構成要素」の「■変数定義」の項を参照してください。

■ データ型 (Type)

変数のデータ型を以下の中から選択します。

BOOL、INT、UINT、DINT、UDINT、LINT、ULINT、WORD、DWORD、LWORD、REAL、LREAL、TIMER、COUNTER、STRING

変数のデータ型の説明は、2-1-2 項「ファンクションブロックの構成要素」の「■変数定義」の項を参照してください。

■ AT 指定 (実アドレスへの割り当て)

変数の使用アドレスを、システムによる自動割当てでなく、特定の I/O メモリ実アドレスとすることも可能です。その場合、ユーザがこの属性に I/O メモリ実アドレスを入力します。

ただし、内部変数 (Internals) の場合のみ可能です。

AT 指定を使用すると、ファンクションブロックの汎用性が失われるため、AT 指定は、必要最小限にしてください。

AT 指定の詳細は、2-1-2 項「ファンクションブロックの構成要素」の「■変数定義」の項を参照してください。AT 指定の使い方は、2-4 項「ファンクションブロック使用のガイドライン」の 2-4-3 項「内部変数の AT 指定」を参照してください。

■ 配列指定

変数を、同一属性のデータの、1 つの集合体 (配列) として扱うことも可能です。その場合、ユーザは、この属性に、配列化する最大要素数を入力します。

ただし、内部変数 (Internals)、入出力変数 (In Out) の場合のみ可能です。

CX-Programmer Ver.5.0 以降では、1 次元配列のみが可能です。

- ・ 設定方法：「詳細」ボタンをクリックして、配列設定フィールドの配列変数をチェックして、最大要素数を入力します。
- ・ ファンクションブロック定義のアルゴリズム内で変数名を記述するときは、変数名の後に、添字を半角 [] で囲んで指定します。

→配列指定の詳細は、本編の第 2 章「ファンクションブロックの仕様」の 2-1-2 項「ファンクションブロックの構成要素」の「■変数定義」を参照ください。

1-3 変数とは

1-3-4 変数の属性と変数種類の関係

■ 初期値 (Initial Value)

インスタンスが初めて実行されるときに、変数は、いったんここに設定された値となります。その後、インスタンスの実行によって変更可能となります。

例えば、接点 (BOOL) は、1 (TRUE) または 0 (FALSE) を選択します。チャンネル (WORD) は、0~65535 (0000~FFFFHex) を入力します。

この属性を設定しない場合、例えば、接点 (BOOL) は、0 (FALSE) となります。チャンネル (WORD) は、0000Hex (0) となります。

■ 電断または運転開始時保持 (Retained) :

PLC の電源 OFF→ON 時または運転開始時に、変数のデータを保持したいとき、保持を有効にします。

・設定方法：[保持] チェックボックスをチェックします。

■ サイズ (Size) :

変数が STRING 型の場合に、文字列を格納するためのサイズを設定することができます。文字数は 1~255 の範囲で設定できます。

1-3-4 変数の属性と変数種類の関係

変数の属性の設定必要/可能/不可は、変数の種類 (Usage) によって、以下のように異なります。

○：設定必要、△：設定可能、×：設定不可

変数の属性 \ 変数の種類	内部変数 (Internals)	入力変数 (Inputs)	出力変数 (Outputs)	入出力変数 (In Out)
変数名 (Name)	○	○	○	○
データ型 (Type)	○	○	○	○
AT (実アドレスへの割り当て)	△	×	×	×
配列指定	○	×	×	○
初期値 (Initial Value)	△	× (注 1)	△	×
電断または運転開始時保持 (Retained)	△	× (注 1)	△	×
サイズ (Size)	△ (注 2)	×	×	×

注 1：入力パラメータの値になります。

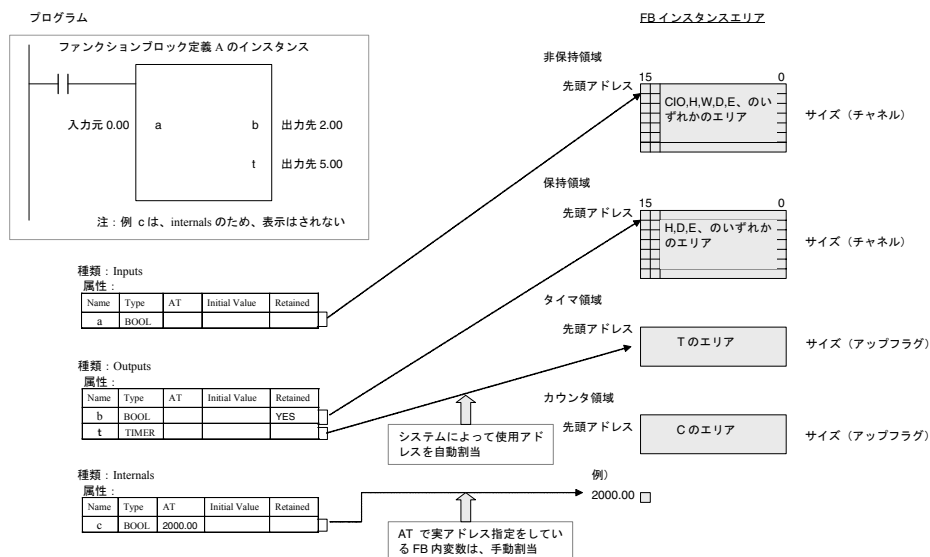
注 2：STRING 型のみ有効となります。

1-3 変数とは

1-3-5 変数の使用アドレスの内部割当

1 1-3-5 変数の使用アドレスの内部割当

ファンクションブロック定義からそのインスタンスを生成すると、ファンクションブロック定義で登録したすべての変数（AT 指定をしている変数以外）に、使用アドレスがシステム（CX-Programmer）によって内部的に割り当てられます。



■ 変数の内部割当範囲の設定

システムによって内部的に自動割当をする領域（FB インスタンスエリア）をどこにするかを、ユーザが設定します。変数は、ユーザが設定した FB インスタンスエリアのいずれかに、システムによって自動割り当てされます。

● 設定方法

[PLC] メニューから [ファンクションブロック/SFC メモリ] | [ファンクションブロック/SFC アドレス割付] を選択します。[ファンクションブロック/SFC アドレス割付] ダイアログボックスにて設定します。

● FB インスタンスエリア

・ CJ2 シリーズ CPU ユニットのの場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	サイズ	
保持以外	H512	H1407	896	CIO、WR、HR、DM、EM ^{*1}
保持	H1408	H1535	128	HR、DM、EM ^{*1}
タイマ	T3072	T4095	1024	TIM
カウンタ	C3072	C4095	1024	CNT

*1. 以下の EM バンクを指定した場合、強制セット／リセットが可能となります。

形 CJ2H-CPU64/65(-EIP)	EM バンク 3
形 CJ2H-CPU66(-EIP)	EM バンク 6～9
形 CJ2H-CPU67(-EIP)	EM バンク 7～E
形 CJ2H-CPU68(-EIP)	EM バンク 11～18

・ CS/CJ シリーズ CPU ユニット Ver.3.0 以降、NSJ の場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	サイズ	
保持以外	H512	H1407	896	CIO、WR、HR、DM、EM
保持	H1408	H1535	128	HR、DM、EM
タイマ	T3072	T4095	1024	TIM
カウンタ	C3072	C4095	1024	CNT

・ FQM1 の場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	サイズ	
保持以外	5000	5999	1000	CIO、WR、DM
保持	なし			
タイマ	T206	T255	50	TIM
カウンタ	C206	C255	50	CNT

・ CP1 シリーズ CPU ユニットのの場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	サイズ	
保持以外	H512	H1407	896	CIO、WR、HR、DM ^{*1}
保持	H1408	H1535	128	HR、DM ^{*1}
タイマ	T3072	T4095	1024	TIM
カウンタ	C3072	C4095	1024	CNT

*1. CP1L-L の DM エリアは以下のようになっています。

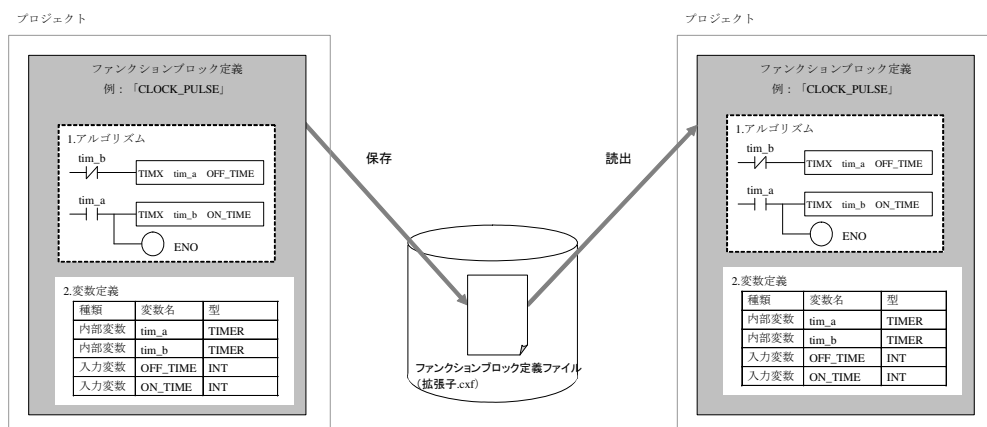
アドレス	CP1L-L
D0000～D9999	あり
D10000～D31999	存在しない
D32000～D32767	あり

・ CP2E シリーズ CPU ユニットのの場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	サイズ	
保持以外	H512	H1407	896	CIO、WR、HR、DM
保持	H1408	H1535	128	HR、DM
タイマ	T256	T511	256	TIM
カウンタ	C256	C511	256	CNT

1-4 ファンクションブロック定義のライブラリ化

CX-Programmer では、作成した1つのファンクションブロック定義を1つのファイル（「ファンクションブロック定義ファイル」と呼ぶ。拡張子.cxf）に保存して、他のプロジェクト（PLC）で再利用することができます。



1-5 使用手順の概要

1

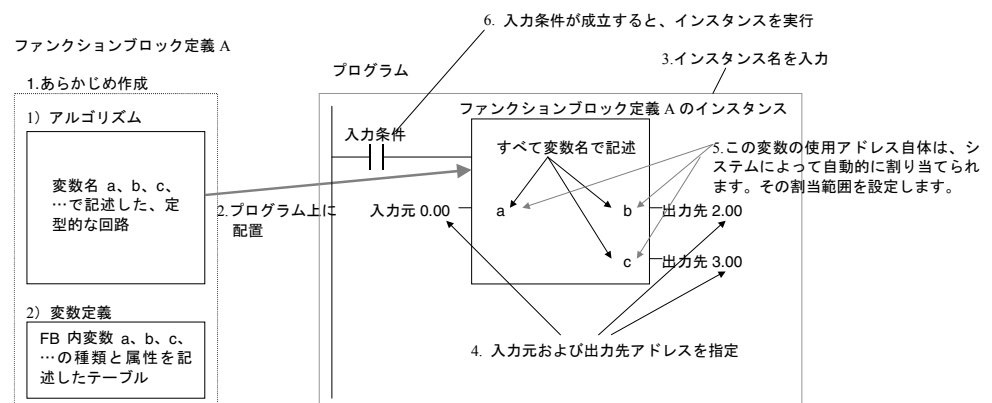
ファンクションブロックの概要

あらかじめ「ファンクションブロック定義」を作成した上で、アルゴリズムのコピーを生成し（「インスタンス」と呼ぶ）、それを実行時に呼び出す形で使用します。また、作成した「ファンクションブロック定義」をファイルに保存して、他のプロジェクト（PLC）で再利用します。

1-5-1 ファンクションブロックの作成から実行まで

ファンクションブロックを作成し実行するまでの手順の概要は、以下のとおりです。

1. あらかじめ、ファンクションブロック定義（アルゴリズムと変数定義）をラダーまたは ST 言語で作成します。
または、あらかじめ用意された FB ライブラリファイルを挿入します。
注1：アルゴリズムはすべて変数名で記述します。
注2：ファンクションブロックをラダー言語で記述する場合、既存の CX-Programmer で作成したプロジェクトファイルを CX-Programmer Ver.5.0 以降で読み出して、その中の回路をコピー（カット）＆ペーストおよび編集して、作成することも可能です。
注3：既存のラダープログラムを流用する場合、[ファンクションブロック（ラダー）生成] より、自動でファンクションブロック化することができます。
2. プログラム作成時に、作成済みのファンクションブロック定義を配置します。これにより、ファンクションブロックのコピー（「インスタンス」と呼ぶ）を生成します。
3. それに名称（「インスタンス名」と呼ぶ）をつけます。
4. インスタンスに対する受け渡しのパラメータとして、変数の入力元アドレス（または値）、出力先アドレス（または値）を設定します。
5. 作成したインスタンスを選択して、[PLC] メニューから [ファンクションブロックメモリ] | [ファンクションブロック アドレス割付] を選択して、変数の内部割当範囲を設定します。
6. プログラムを実機 PLC へ転送します。
7. PLC を運転開始して、入力条件が成立すると、インスタンスを呼び出す形で、その処理を実行します。



1-5 使用手順の概要

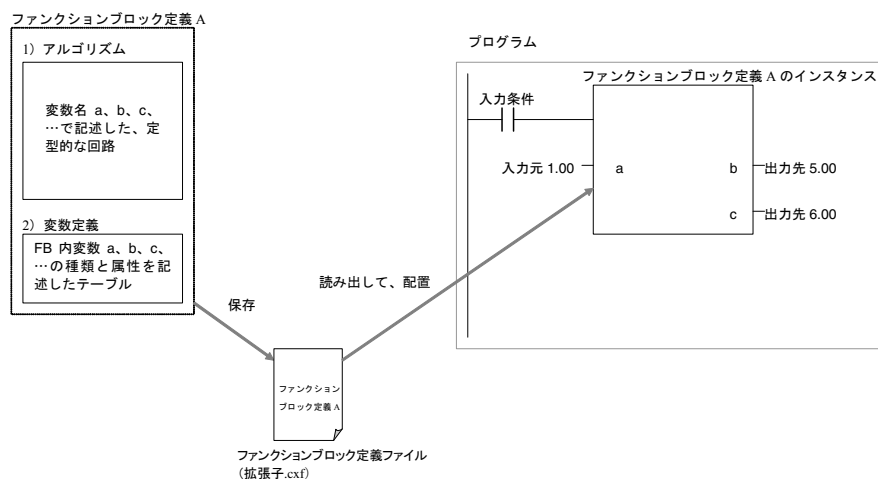
1-5-2 ファンクションブロックの再利用

1

1-5-2 ファンクションブロックの再利用

作成したファンクションブロック定義は、以下の操作によって、1 ファンクションブロック定義=1 ファイルとして保存して、他の PLC 用プログラムで再利用できます。

1. 保存したいファンクションブロックを選択し、ファンクションブロック定義ファイル (*.cxf) として保存します。
2. 他の PLC 用のプロジェクト (PLC) を開いて、保存してあるファンクションブロック定義ファイル (拡張子.cxf) を開いて読み込みます。
3. プログラム作成時に、ファンクションブロック定義を配置します



参 考

CX-Programmer では、ファンクションブロック定義ごとに、コンパイル処理をしてプログラムチェックをかけることができます。ファイル保存前または再利用時に、ファンクションブロック定義ごとにコンパイル (プログラムチェック) をかけることをお勧めします。

ファンクションブロック編

第2章 ファンクションブ ロックの仕様

2-1 ファンクションブロックの仕様

2-1-1 ファンクションブロックの仕様

項目	説明
ファンクションブロック定義数	1 台の PLC (CPU ユニット) あたり、 CJ2H の場合： ・形 CJ2H-CPU6□(-EIP)：最大 2048 CJ2M の場合： ・形 CJ2M-CPU□1/□2/□3：最大 256 ・形 CJ2M-CPU□4/□5：最大 2048 CS1-H /CJ1-H の場合： ・CPU ユニット形式末尾-CPU67H-R/66H-R/65H-R/64H-R/67H/66H/65H/45H/64H/44H：最大 1024 ・CPU ユニット形式末尾-CPU63H/43H/42H：最大 128 CS1D-H の場合： ・形 CS1D-CPU68HA/67HA：最大 1024 CS1D-S の場合： ・形 CS1D-CPU67SA/44SA：最大 1024 CJ1M の場合： ・形 CJ1M- CPU23/22/21/13/12/11：最大 128 CP1H の場合： ・形 CP1H-XA/X/Y：最大 128 CP1L の場合： ・形 CP1L-M/L：最大 128 CP1L-E の場合： ・形 CP1L-EM/EL：最大 128 CP2E の場合： ・形 CP2E-E/S：最大 64 CP2E-N の場合： ・形 CP2E-N：最大 64 NSJ の場合：最大 1024 FQM1 の場合： ・形 FQM1-CM002、形 FQM1-MMA22、形 FQM1-MMP22：最大 128
インスタンス数	1 台の PLC (CPU ユニット) あたり、 CJ2H の場合： ・形 CJ2H-CPU6□(-EIP)：最大 2048 CJ2M の場合： ・形 CJ2M-CPU□1/□2/□3：最大 256 ・形 CJ2M-CPU□4/□5：最大 2048 CS1-H /CJ1-H の場合： ・CPU ユニット形式末尾-CPU67H-R/66H-R/65H-R/64H-R/67H/66H/65H/45H/64H/44H：最大 2048 ・CPU ユニット形式末尾-CPU63H/43H/42H：最大 256 CS1D-H の場合： ・形 CS1D-CPU68HA/67HA：最大 2048 CS1D-S の場合： ・形 CS1D-CPU67SA/44SA：最大 2048 CJ1M の場合： ・形 CJ1M- CPU23/22/21/13/12/11：最大 256 CP1H の場合： ・形 CP1H-XA/X/Y：最大 256 CP1L の場合 ・形 CP1L-M/L：最大 256 CP1L-E の場合 ・形 CP1L-EM/EL：最大 256 CP2E の場合： ・形 CP2E-E/S：最大 128 CP2E-N の場合： ・形 CP2E-N：最大 128 NSJ の場合：最大 2048 FQM1 の場合： ・形 FQM1-CM002、形 FQM1-MMA22、形 FQM1-MMP22：最大 256

2-1 ファンクションブロックの仕様

2-1-1 ファンクションブロックの仕様

項目	説明
インスタンスのネスティング数	CX-Programmer Ver.5.0：ネスティングは不可 CX-Programmer Ver.6.0 以降：最大 8 段（プログラム上からのインスタンスの呼び出し自体を 1 段として含む）
ファンクションブロックで利用できる変数の数（内部変数、外部変数、EN、ENO を除く）	ファンクションブロック定義 1 つにつき ・入力変数 + 入出力変数：最大 64 個 ・出力変数 + 入出力変数：最大 64 個 ・入出力変数：最大 16 個

2-1-2 ファンクションブロックの構成要素

ユーザがファンクションブロック定義に関して記述が必要な項目は以下のとおりです。

項目	説明
ファンクションブロック定義名	ファンクションブロック定義の名称。
言語種別	ファンクションブロック定義の言語の種別。 ラダーまたは ST を選択します。
変数定義	ファンクションブロック定義に対する引数や戻り値など、実行に必要なデータの型 ・変数の種類 ・変数名 ・変数のデータ型 ・変数の初期値
アルゴリズム	ラダーまたは ST で、ロジックを記載します。 ・変数を用いてロジックを記載する ・ただし、定数は、変数に登録せず直接入力する
コメント	コメントを持つことができます。

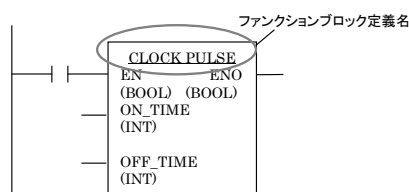
■ ファンクションブロック定義名

ファンクションブロック定義の名称です。

半角または全角文字を入力できます。最大全角 32 文字（半角 64 文字）まで。

禁則文字はありません。

初期値は「ファンクションブロック*」です（*は、1 からの連番）。



■ 言語種別

ラダー、または ST 言語を選択します。

注 1：ST 言語の仕様については、ストラクチャードテキスト編 第 5 章「ST 言語の仕様」を参照してください。

注 2：ネスティング時のラダー/ST 言語の呼び出しの組み合わせは自由です。（Ver.6.0 以降）

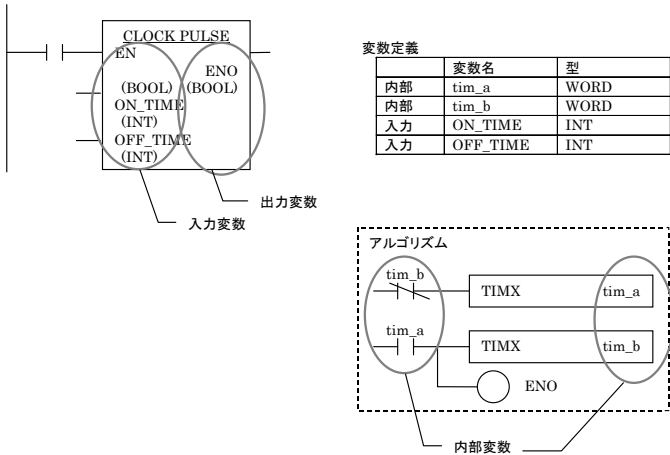
■ 変数定義

引数やファンクションブロック定義内で使用する変数の定義を行います。

● 変数名

- ・半角または全角文字を入力できます。
- ・最大全角 15000 文字（半角 30000 文字）まで。
- ・以下の [] 内に示す、文字およびスペースは使用できません。
[!], [“], [#], [\$], [%], [&], [‘], [(], [)], [=], [-], [~], [^],
[¥], [|], [’], [@], [{], [], [+], [;], [*], [:], [}], []], [<],
[,], [>], [.] , [?], [/]
- ・数字 (0~9) からはじめることは不可。
- ・以下の I/O メモリアドレスを示す文字列を入力することは不可。
A、W、H（または HR）、D（または DM）、E（または EM）、T（または TIM）、
C（または CNT）を先頭としてその後に数値（チャンネル番号）

● 変数の表記

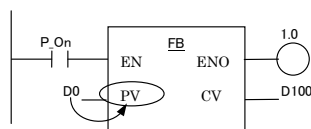


● 変数の種類

種類*3	定義	次回の実行時	図示表示	使用数	AT 設定	配列 指定	電断または運転開始時保持 設定	デフォルトで 必ず作成される変数
入力変数 (Inputs)	インスタンス に対する引数	入力パラメータ の値になります	インスタ ンスの左 側に表示 されます	1 ファンクシ ョンブロック あたり最大 64 個 (EN を除く)	不可	不可	可能*2	"EN" (Enable) : 入力条件を受け る変数
出力変数 (Outputs)	インスタンス からの戻り値	次回の実行時ま で値が保持され ます	インスタ ンスの右 側に表示 されます	1 ファンクシ ョンブロック あたり最大 64 個 (ENO を除 く)	不可	不可	可能	"ENO" (Enable Output) : ファン クションブ ロックの実行 状態を出力す る変数
入出力変数 (In Out)	インスタンス とデータをアド レスで受け 渡す変数	外部パラメータ の値になります	インスタ ンスの左側と 右側に対で 表示されま す	1 ファンクシ ョンブロックあ たり最大 16 個	不可	可能	不可	なし
内部変数 (Internals)	インスタンス の内部だけで 使用する変数	次回の実行時ま で値が保持され ます	表示され ません	制限なし	可能、	可能	可能	なし
外部変数 (Externals) *1	CX-Programmer にあらかじめ 登録されてい るシステム定 義変数、または ユーザ定義の グローバル変 数	外部で登録され た変数の値にな ります	表示され ません	制限なし	不可	不可	不可	コンディショ ンフラグ、一部の 特殊補助リレー など、 CX-Programmer にあらかじめ登 録されているシ ステム定義変数

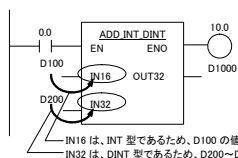
*1：外部変数（Externals）の詳細は、付-1の「ファンクションブロックで使用するシステム定義の外部変数一覧」を参照。
*2：入力パラメータの値となります。
*3：構造体変数、TIMER 型/COUNTER 型変数は、以下の変数でのみ使用可能です。
構造体変数：内部変数、入出力変数、外部変数
TIMER 型/COUNTER 型変数：内部変数、外部変数

インスタンスへの外部からの引数です。インスタンスの左側に表示されます。
入力元として指定されたパラメータの、インスタンス呼出直前の値（データ）が、入力変数に転送されます。

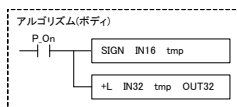


入力に指定したパラメータの値(P0 の値)がインスタンスの入力変数(PV)に転送される。

例)



- IN16 は、INT 型であるため、D100 の値が、入力変数となる。
- IN32 は、DINT 型であるため、D200～D201 の値が入力変数となる。



変数定義	変数名	データ型
内部	tmp	DINT
入力	EN	BOOL
入力	IN16	INT
入力	IN32	DINT
出力	ENO	BOOL
出力	OUT32	DINT

[illegible]

注1：入力変数と出力変数は、同一変数名とすることはできません。外部のパラメータから受け取った値を、ファンクションブロック内で変更してパラメータに戻したい場合には、入出力変数を使用してください。

注 2：パラメータから入力変数へ、値を読み出すタイミングは、インスタンス実行時のアルゴリズム処理前です。したがって、アルゴリズムの中で、パラメータから入力変数へ値を読み出すことはできません。アルゴリズムの中で（その実行サイクル内で）値を読出したい場合は、パラメータから値を読み出さず、内部変数として AT 指定（実アドレスへの割り当て）をしてください。または、外部変数としてグローバル変数を参照してください。

入力変数に、初期値を設定することはできますが、入力パラメータの値の方が有効となります（入力変数 EN へのパラメータが ON となってインスタンスが実行されたとき、入力パラメータの値がセットされます）。

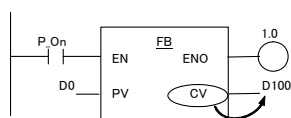
注：CX-Programmerでは、入力パラメータの設定を、省略することはできません。

この変数は、デフォルトで入力変数として生成されます。

入力変数EN へのパラメータがON のとき、インスタンスが実行されます。

インスタンスから外部への戻り値です。インスタンスの右側に表示されます。

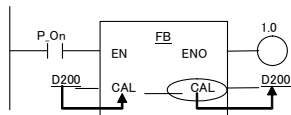
インスタンス実行後の値が、出力先として指定されたパラメータへ転送されます。



出力変数 CV)の値が、出力先に指定したパラメータ D100 に転送される

インスタンス外部とのデータの受け渡しに、アドレスを使用する変数です。インスタンスの左側と右側に対で表示されます。

したがって、インスタンスの実行後の入出力変数の値は、システムで内部的に入出力変数に割り当てられたアドレスに保存されるわけではなく、入出力変数との受け渡し用のパラメータのアドレス（以降のデータサイズ分）に、保存される点に注意が必要です。



入出力変数 CAL にアドレス D200 が渡り、FB 内で、D200 以降のデータサイズ分を変更処理し、変更内容を FB 外部に反映

インスタンス内部で使用する変数です。

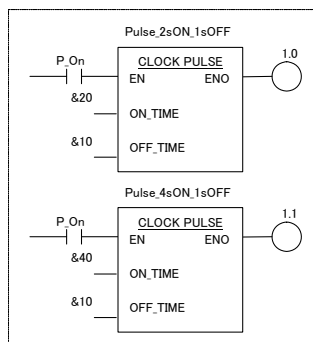
各インスタンスは、この変数を内部に隠ぺい化します。外部から参照することができません。インスタンスには表示されません。

内部変数は、次のインスタンス実行時まで値を保持します（つまり、EN が OFF のとき、内部変数の値は、保持されたままとなります）。

したがって、同じファンクションブロックのインスタンスで、同じ入出力パラメータでも、同じ結果となるとはかぎりません。

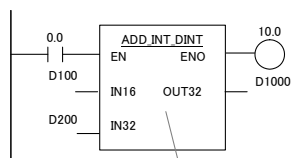
例) インスタンス「Pulse 2sON 1sOFF」が持つ内部変数tim aとインスタンス

「Pulse_4sON_1sOFF」が持つ内部変数 tim_a は別のものであり、お互いの値を参照することはできません。

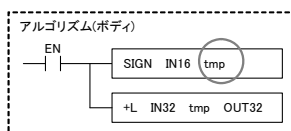


変数定義		
	変数名	データ型
内部	tim_a	WORD
内部	tim_b	WORD
入力	ON_TIME	INT
入力	OFF_TIME	INT

名称	データ型	AT	初期値	保持	コメント
tmp	UDINT		0		
内部変数					
入力変数		出力変数		外部変数	



内部変数 tmp は、表示されない。



	変数名	データ型
内部	tmp	DINT
入力	EN	BOOL
入力	IN16	INT
入力	IN32	DINT
出力	ENO	BOOL
出力	OUT32	DINT

・電断/運転開始時保持：

内部変数は、前回インスタンスが呼び出されたときの値を保持しています。
さらに、電源 OFF→ON 時または運転開始時（「プログラム」から「モニタ」または「運転」モードへ変更時）にも、値を保持するか否かを選択することができます。
・「保持」（変数の属性の「保持」チェックボックスをチェック）に設定された変数は、電源 ON/OFF 時、および運転開始時に、保持されます。
ただし、CPU ユニット側にバッテリーがない場合は、不定となります。

	条件	常時
保持に設定された変数	運転開始時	保持
	電源 ON/OFF 時	保持

・「非保持」（変数の属性の「保持」チェックボックスをチェックせず）に設定された変数は、電源 ON/OFF 時、および運転開始時に、保持されません。
なお、「非保持」に設定された変数は、I/O メモリ保持フラグ（A50012）を ON のときは、運転開始時に保持することができます（*1）。さらに、PLC システム設定「電源 ON 時に、I/O メモリ保持フラグを保持する」に設定した場合は、電源 ON/OFF 時にも、保持できます。
ただし、CPU ユニット側にバッテリーがない場合は初期化され、0 となります。

	条件	I/O メモリ保持フラグ（A50012）		
		OFF	ON（*1）	
			PLC システム設定「電源 ON 時に、I/O メモリ保持フラグを保持する」	PLC システム設定「電源 ON 時に、I/O メモリ保持フラグを保持しない」
非保持に設定された変数	運転開始時	保持しない	保持	保持
	電源 ON/OFF 時	保持しない	保持	保持しない

*1：本 I/O メモリ保持フラグ機能は、既存機種との互換性のために存在します。値を保持したい場合は、本機能でなく、変数の属性の「保持」をチェックしてください。

・初期値：

非保持指定の内部変数には、初期値を設定することができます。保持指定の内部変数には、初期値を指定することはできません。
初期値の設定がされていない非保持指定の内部変数には、ゼロ（0）がセットされます。
ただし、I/O メモリ保持フラグ（A50012）が ON に設定されている場合は、初期値はセットされません。

特殊補助リレー		初期値
I/O メモリ保持（A50012）	ON	セットされない
	OFF	セットされる

・外部変数

CX-Programmer にあらかじめ登録されているシステム定義変数、またはグローバル変数テーブルにユーザ定義されている変数を外部参照する変数です。

- ・システム定義変数については、付-1 の「ファンクションブロックで使用するシステム定義の外部変数一覧」を参照してください。
- ・グローバル変数テーブルにユーザ定義されている変数を外部参照するには、変数名およびデータ型が同一の変数を、外部変数に登録していることが必要です。
ただし、ネットワーク変数にユーザ定義された変数を外部参照することはできません。

● 変数の属性

・変数名 (Name)

ファンクションブロック内で変数を識別するための名称。最大全角 15000 文字。

他のファンクションブロックで同一名を使用してもかまいません。

注：AT 指定 (実アドレスへの割り当て) をしている場合でも、変数名は設定が必要です。

・変数のデータ型

以下の型を使用することができます。

データ型	内容	サイズ	入力変数	出力変数	入出力変数	内部変数	外部変数
BOOL	ビットデータ	1 ビット	○	○	○	○	○
INT	整数	16 ビット	○	○	○	○	○
UINT	無符号整数	16 ビット	○	○	○	○	○
DINT	倍精度整数	32 ビット	○	○	○	○	○
UDINT	無符号倍精度整数	32 ビット	○	○	○	○	○
LINT	8 バイト整数	64 ビット	○	○	○	○	○
ULINT	無符号 8 バイト整数	64 ビット	○	○	○	○	○
WORD	16 ビットデータ	16 ビット	○	○	○	○	○
DWORD	32 ビットデータ	32 ビット	○	○	○	○	○
LWORD	64 ビットデータ	64 ビット	○	○	○	○	○
REAL	実数	32 ビット	○	○	○	○	○
LREAL	倍精度実数	64 ビット	○	○	○	○	○
TIMER *1	タイマ	アップフラグ： 1 ビット 現在値：16 ビット	×	×	×	○	○
COUNTER *2	カウンタ	アップフラグ： 1 ビット 現在値：16 ビット	×	×	×	○	○
STRING	文字列データ	可変	×	×	○	○	×
構造体データ型	ユーザ定義のデータ型	可変	×	×	○	○	○

*1：タイマ系命令 (TIM、TIMH など) のタイマ番号 (0~4095) のオペランドに変数を記述するとき、データ型を TIMER とします。この変数を、他の命令で使用したときは、タイムアップフラグ (1 ビット) またはタイマ現在値 (16 ビット) となります (どちらの指定となるかは、命令のオペランドによります)。

*2：カウンタ系命令 (CNT、CNTR など) のカウンタ番号 (0~4095) のオペランドに変数を記述するとき、データ型を COUNTER とします。この変数を、他の命令で使用したときは、カウントアップフラグ (1 ビット) またはカウンタ現在値 (16 ビット) となります (どちらの指定となるかは、命令のオペランドによります)。

・AT 指定 (実アドレスへの割り当て)

内部変数 (Internals) の場合のみ、変数の使用アドレスを、システムによる自動割り当てなく、特定の I/O メモリ実アドレスとすることが可能です。この属性に I/O メモリ実アドレスをユーザが入力します。ただし、その場合でも変数名は設定が必要です。

注 1：AT 指定は、内部変数に対してのみ可能です。

注2：AT 指定できる I/O メモリは、CIO（チャネル I/O）、A（特殊補助リレー）、D（データメモリ）、E（拡張データメモリ）、H（保持リレー）、W（内部補助リレー）のみです。

以下の I/O メモリは、AT 指定できません。

- ・インデックスレジスタ／データレジスタ（間接指定、直接指定とも）
- ・DM/EM 間接指定（@：BIN モード、*：BCD モードとも）

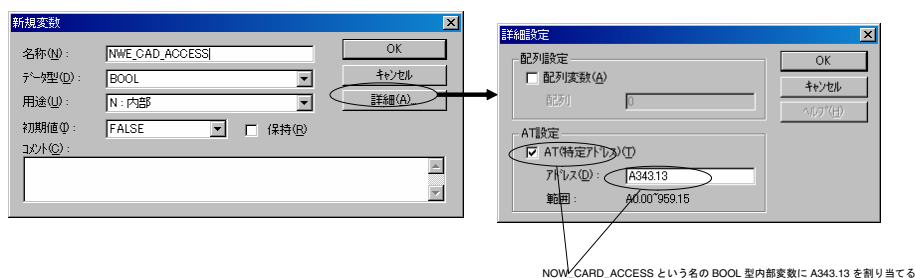
注3：以下のような場合、AT 指定を使用します。

- ・基本 I/O ユニットまたは高機能ユニットの割付アドレス
- ・あらかじめ外部変数に登録されている以外の特殊補助リレー
- ・ネットワーク上の他ノードの PLC のアドレス

例)

ファンクションブロック定義内でファイルメモリ読出命令（FREAD）を使用して、特殊補助リレーの「ファイルメモリ実行中フラグ」（A34313）を参照したいとき、内部変数に AT 指定をします。

内部変数に AT 指定をして、アドレス A34313 を割り当てることで、特殊補助リレーの「ファイルメモリ実行中フラグ」を参照します。



ただし、本機能を使用すると、ファンクションブロックの汎用性が失われるため、必要最小限に使用すべきです。

・配列（ARRAY）指定

内部変数（Internals）/入出力変数（In Out）の場合のみ、変数を配列*1にすることも可能です。

*1：CX-Programmer では、1 次元配列のみが可能です。

これにより、1 つの変数登録で、複数の同一属性の変数を使用できます。データを集合体として扱うときに有効です。

- ・配列の要素数は、内部変数の場合 1 ～ 最大 32000 です。入出力変数の場合はデータ型によって下表のようになります。

データ型	要素数
BOOL	2048
INT / UINT / WORD	2048
DINT / UDINT / DWORD	1024
LINT / ULINT / LWORD	512

- ・配列指定は、内部変数/入出力変数に対してのみ可能です。
- ・STRING 型を除く、すべてのデータ型に対して配列指定が可能です。
- ・ファンクションブロック定義のアルゴリズム内で変数名を記述するときは、変数名の後に、添字を半角 [] で囲んで指定します。
- ・添字の指定方法（例：a[] の、[] の中の指定方法）には、以下の 3 つの方法があります。

- 1) 添字を直接番号で指定する方法（ラダーまたは ST 言語）（例：a[2]）

- 2) 添字を変数で指定する方法（ラダーまたは ST 言語）（例 a[n]。n は変数）
注：変数のデータ型は、INT、DINT、LINT、UINT、UDINT、ULINT が可能です。
- 3) 添字を式で指定する方法（ST 言語のみ可能）（例：a[b+c]。b、c は変数）
注：式は、四則演算（+、-、*、/）が可能です。

参 考 配列とは、同じ型のデータをもつ複数個のデータの集まりです。添字付きの同一変数名で、各配列要素を指定します。（添字は配列中の要素の位置を示します。）
とくに 1 次元配列とは、添字の数が 1 つの配列を指します。

例) 変数名 **SCL** で、配列 (**ARRAY**) 指定をして、配列要素数を **10** とした場合次の **10** 個の変数を使用できます。
SCL[0]、SCL[1]、SCL[2]、SCL[3]、SCL[4]、SCL[5]、SCL[6]、SCL[7]、SCL[8]、SCL[9]

SCL

0	WORD 型
1	WORD 型
2	WORD 型
3	WORD 型
4	WORD 型
5	WORD 型
6	WORD 型
7	WORD 型
8	WORD 型
9	WORD 型

← このデータをアクセスしたい場合、SCL[3]を指定します。

新規変数

名称(N): SCL

データ型(D): WORD

用途(U): N: 内部

初期値(V): 0

コメント(C):

OK

キャンセル

詳細(D)

詳細設定

配列設定

☒ 配列変数(A)

配列: 10

AT設定

☐ AT(特定アドレス) (I)

アドレス(A):

詳細

OK

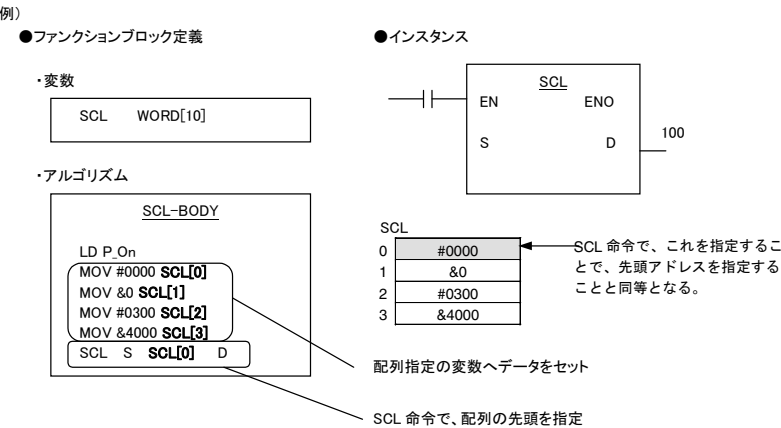
キャンセル

ヘルプ(H)

SCL という変数を、配列指定して、要素数を 0~9 とする

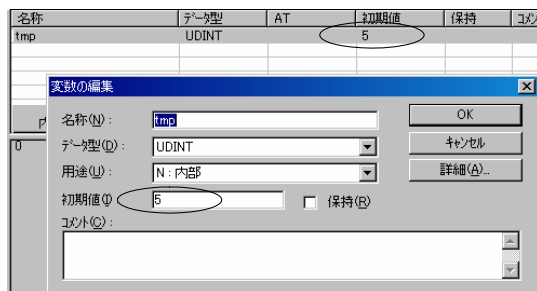
お願い

命令のオペランドが、複数ワード（CH）の先頭アドレス（または最終アドレス）を指定している場合で、ファンクションブロックの汎用性（再利用性）の観点から、そのオペランドを内部変数として AT 指定できない、かつ外部変数指定できない場合、配列指定をします。
入出力変数の配列指定を使用するときは、入力パラメータにその先頭アドレスを設定します（CX-Programmer Ver.7.0 以降のみ可能）。
内部変数の配列指定を使用するときは、必要サイズの要素数の、配列指定を用意し、各配列要素にデータをセットした上で、配列型変数の先頭（または最終）をオペランドに指定します。



・変数の初期値

インスタンスが初めて実行される時に、内部変数、出力変数には初期値をセットすることができます。初期値の詳細は、各入力変数、内部変数、出力変数の初期値の項を参照。



・電断または運転開始時保持（Retained）

内部変数（Internals）の場合に、PLCの電源OFF→ON時または運転開始時に、変数のデータを保持させることができます。「値を保持する」を指定した場合は、内部的には電断/運転開始時保持領域に割り付けられます。

■ アルゴリズム

既に定義した変数を用いてロジックを記述します。

● オペランド入力の制限

- ・ファンクションブロック内では、命令のオペランドに、アドレスを直接入力することはできません。アドレスを直接入力しても、変数名として扱われます。
注：例外的に、インデックスレジスタ（間接指定、直接指定）／データレジスタは、変数ではなく、アドレスを直接入力します。
- ・定数は、直接、命令のオペランドに入力します。
 - －ラダー言語の場合：10進数は、&の後に数値を入力します。
16進数は、#の後に数値を入力します。
 - －ST言語の場合：10進数は、そのまま数値を入力します。
16進数は、16#の後に数値を入力します。

■ プロパティ

コメント

日本語も使用可能であり、最大全角 15000 文字まで。

2-2 ファンクションブロックで使用するデータ型

2

仕様

2-2-1 基本データ型

データ型	内容	ビット	値の範囲
BOOL	ブール型	1	0 (FALSE)、1 (TRUE)
INT	整数型	16	-32768 ~ +32767
DINT	倍精度整数型	32	-2147483648 ~ +2147483647
LINT	4 倍精度整数型	64	-9223372036854775808 ~ +9223372036854775807
UINT	符号なし整数型	16	&0 ~ 65535
UDINT	符号なし倍精度整数型	32	&0 ~ 4294967295
ULINT	符号なし 4 倍精度整数型	64	&0 ~ 18446744073709551615
REAL	実数型	32	-3.402823×10 ³⁸ ~-1.175494×10 ⁻³⁸ 、0、+1.175494×10 ⁻³⁸ ~+3.402823×10 ³⁸
LREAL	倍精度実数型	64	-1.79769313486232×10 ³⁰⁸ ~-2.22507385850720×10 ⁻³⁰⁸ 、0、+2.22507385850720×10 ⁻³⁰⁸ ~+1.79769313486232×10 ³⁰⁸
WORD	長さ 16 のビット列型	16	#0000~FFFF または&0 ~ 65535
DWORD	長さ 32 のビット列型	32	#00000000~FFFFFFFF または&0 ~ 4294967295
LWORD	長さ 64 のビット列型	64	#0000000000000000~FFFFFFFFFFFFFFFF または&0 ~ 18446744073709551615
STRING	文字列型	可変	文字数 1~255 の ASCII コード
TIMER	タイマ型	1 および 16	タイマ番号：0~4095 タイムアップフラグ：0、1 タイマ現在値：0~9999 (BCD 方式)、0~65535 (BIN 方式)
COUNTER	カウンタ型	1 および 16	カウンタ番号：0~4095 カウントアップフラグ：0、1 カウンタ現在値：0~9999 (BCD 方式)、0~65535 (BIN 方式)
FUNCTION BLOCK	ファンクションブロックインスタンス	—	—

2-2-2 派生データ型

データ型	内容
配列データ型	1 次元配列。最大要素数: 32000
構造体データ型	ユーザ定義のデータ型

2-3 インスタンスの仕様

2-3-1 インスタンスの構成要素

ユーザがインスタンスに関して記述が必要な項目は以下のとおりです。

項目	説明
インスタンス名称	インスタンスの名称
言語種別 変数定義	ファンクションブロック定義を継承する (ファンクションブロック定義と同じ)
FB インスタンスエリア	変数の使用アドレス領域を設定する。
コメント	インスタンスごとにコメントを持つことができる

■ インスタンス名称

インスタンスの名称です。

半角または全角文字を入力できます。

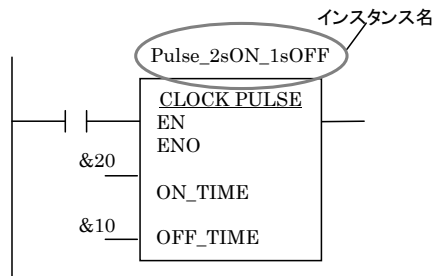
- ・最大全角 15000 文字（半角 30000 文字）まで。
- ・以下の [] 内に示す、文字およびスペースは使用できません。

[!] , [“] , [#] , [\$] , [%] , [&] , ['] , [(] , [)] , [=] , [-] , [~] , [^] ,
[¥] , [|] , ['] , [@] , [{] , [] , [+], [,], [*], [:], [}], [] , [<] ,
[,] , [>] , [,] , [?] , [/]

- ・数字 (0~9) から始めることは不可。

それ以外の禁則はありません

図示表示では、インスタンス図の上側に表示されます。



■ FB インスタンスエリア

ファンクションブロックを実現する際、システムはインスタンスの内部変数や入力変数、出力変数、または入出力変数を格納する使用エリアを必要とします。このエリアを「FB インスタンスエリア」と呼び、開始アドレスとサイズをユーザが指定します。開始アドレス、サイズともに、1CH 単位で指定することができます。

ユーザプログラム上で本エリアにアクセスする命令が存在すると、CX-Programmer はコンパイル時にエラーを出力します。



・ CJ2 シリーズ CPU ユニットの場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	サイズ	
保持以外	H512	H1407	896	CIO、WR、HR、DM、EM ^{*1}
保持	H1408	H1535	128	HR、DM、EM ^{*1}
タイマ	T3072	T4095	1024	TIM
カウンタ	C3072	C4095	1024	CNT

*1. 以下の EM バンクを指定した場合、強制セット／リセットが可能となります。

形 CJ2H-CPU64/65(-EIP)	EM バンク 3
形 CJ2H-CPU66(-EIP)	EM バンク 6～9
形 CJ2H-CPU67(-EIP)	EM バンク 7～E
形 CJ2H-CPU68(-EIP)	EM バンク 11～18

・ CS/CJ シリーズ CPU ユニット Ver.3.0 以降、NSJ の場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	サイズ	
保持以外	H512	H1407	896	CIO、WR、HR、DM、EM
保持	H1408	H1535	128	HR、DM、EM
タイマ	T3072	T4095	1024	TIM
カウンタ	C3072	C4095	1024	CNT

・ FQM1 の場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	サイズ	
保持以外	5000	5999	1000	CIO、WR、DM
保持	なし			
タイマ	T206	T255	50	TIM
カウンタ	C206	C255	50	CNT

・ CP1 シリーズ CPU ユニットの場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	エリア	
保持以外	H512	H1407	896	CIO、WR、HR、DM ^{*1}
保持	H1408	H1535	128	HR、DM ^{*1}
タイマ	T3072	T4095	1024	TIM
カウンタ	C3072	C4095	1024	CNT

*1. CP1L-L の DM エリアは以下のようにになっています。

アドレス	CP1L-L
D0000～D9999	あり
D10000～D31999	存在しない
D32000～D32767	あり

・ CP2E シリーズ CPU ユニットの場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	エリア	
保持以外	H512	H1407	896	CIO、WR、HR、DM
保持	H1408	H1535	128	HR、DM
タイマ	T256	T511	256	TIM
カウンタ	C256	C511	256	CNT

● FB インスタンスエリアの種類

FB インスタンスエリアには、以下の種類があります。

- ・ CS/CJ シリーズ CPU ユニット Ver.3.0 以降、CP シリーズ、NSJ の場合
- ・ 保持以外（非保持）領域

項目	内容
割付対象変数	電断/運転開始時保持属性を非保持に設定している変数*1
設定可能エリア	H（ファンクションブロック専用保持リレー）、I/O（CIO）、H（保持リレー）、W（内部補助リレー）、D（データメモリ）*2、E（拡張データメモリ）*2*3
設定単位	CH（ワード）単位で設定
デフォルトでの割付エリア	H512～1407CH

*1：データ型（Type）が TIMER および COUNTER を除く

*2：保持領域・非保持領域に、D（データメモリ）または E（拡張データメモリ）を指定した場合でも、ビットアクセスが可能です。

*3：保持領域・非保持領域に、E（拡張データメモリ）を指定した場合、ユーザプログラム上で、同一バンク No. をカレントバンクに指定することはできません。

- ・ 保持領域

項目	内容
割付対象変数	電断/運転開始時保持属性を保持に設定している変数*1
設定可能エリア	H（ファンクションブロック専用保持リレー）、H（保持リレー）、D（データメモリ）*1、E（拡張データメモリ）*2*3
設定単位	CH（ワード）単位で設定
デフォルトでの割付エリア	H1408～1535CH

*1：データ型（Type）が TIMER および COUNTER を除く

*2：保持領域・非保持領域に、D（データメモリ）または E（拡張データメモリ）を指定した場合でも、ビットアクセスが可能です。

*3：保持領域・非保持領域に、E（拡張データメモリ）を指定した場合、ユーザプログラム上で、同一バンク No. をカレントバンクに指定することができません。

- ・ タイマ領域

項目	内容
割付対象変数	データ型（Type）に TIMER を設定している変数
設定可能エリア	T（タイマ）のタイムアップフラグ（1 ビット）またはタイマ現在値（16 ビット）
デフォルトでの割付エリア	T3072～4095 のタイムアップフラグ（1 ビット）またはタイマ現在値（16 ビット）*1

*1：CP2E のデフォルト割付エリアは、T256～511

- ・ カウンタ領域

項目	内容
割付対象変数	データ型（Type）に COUNTER を設定している変数
設定可能エリア	C（カウンタ）のカウントアップフラグ（1 ビット）またはカウンタ現在値（16 ビット）
デフォルトでの割付エリア	C3072～4095 のカウントアップフラグ（1 ビット）またはカウンタ現在値（16 ビット）*1

*1：CP2E のデフォルト割付エリアは、C256～511

● ファンクションブロック専用保持リレー（H512～H1535）

デフォルトで、「非保持」「保持」に設定されている H512～H1535 は、ファンクションブロック専用保持リレーです。

ユーザプログラム用の通常の保持リレーエリア（H000～H511）と異なり、FB インスタンスエリア（変数の内部割当エリア）としてのみ使用可能なエリアです。

- ・内部変数の AT 指定に、指定することはできません。
- ・ユーザプログラム上の命令のオペランドとして、指定することはできません。
 - －ファンクションブロックを作成していない場合は、入力すると赤字で表示されます。
 - －ファンクションブロックを作成している場合は、入力可能ですが、プログラムチェックで、エラーとなります。
- ・保持リレーですが、「保持以外」にこのエリアを指定した場合、電源 ON/OFF、または運転開始時に、値は保持せず、クリアされます。

お願い

FB インスタンスエリアとユーザプログラム上での使用アドレスの重複を避けるため、「保持以外」（非保持領域）および「保持」（保持領域）には、基本的に、ファンクションブロック専用保持リレー（H512～1535）を設定してください。
領域が不足する場合は、ユーザプログラム上で使用していないエリアを設定してください。

・ FQM1 の場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	サイズ	
保持以外	5000	5999	1000	CIO、WR、DM
保持	なし			
タイマ	T206	T255	50	TIM
カウンタ	C206	C255	50	CNT

・ 保持以外（非保持）領域

項目	内容
割付対象変数	電断/運転開始時保持属性を非保持に設定している変数*1
設定可能エリア	I/O（CIO）、W（内部補助リレー）、D（データメモリ）*2、
設定単位	CH（ワード）単位で設定
デフォルトでの割付エリア	5000～5999CH

*1：データ型（Type）が TIMER および COUNTER を除く

*2：非保持領域に、D（データメモリ）を指定した場合でも、ビットアクセスが可能です。

・ 保持領域

なし

・ タイマ領域

項目	内容
割付対象変数	データ型（Type）に TIMER を設定している変数
設定可能エリア	T（タイマ）のタイムアップフラグ（1 ビット）またはタイマ現在値（16 ビット）
デフォルトでの割付エリア	T206～255 のタイムアップフラグ（1 ビット）またはタイマ現在値（16 ビット）

・ カウンタ領域

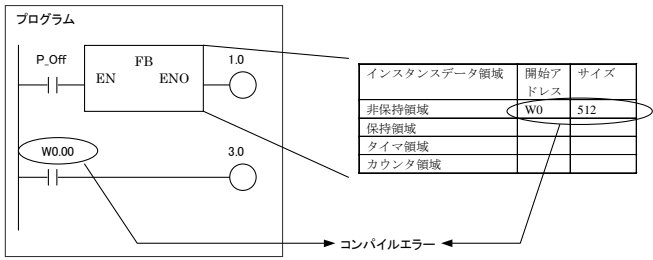
項目	内容
割付対象変数	データ型（Type）に COUNTER を設定している変数
設定可能エリア	C（カウンタ）のカウントアップフラグ（1 ビット）またはカウンタ現在値（16 ビット）
デフォルトでの割付エリア	C206～255 のカウントアップフラグ（1 ビット）またはカウンタ現在値（16 ビット）

● プログラム上で FB インスタンスエリアをアクセスした場合の動作

ユーザプログラム上で「FB インスタンスエリア」にアクセスする命令が存在すると、CX-Programmer は、以下のときに、アウトプットウィンドウの [コンパイル (プログラムチェック)] タブに、エラーを表示します。

- ・オンラインエディットをして書き込みをしようとしたとき (書き込みは不可)
- ・ユーザによるプログラムチェック実行時 ([プログラム] メニューから [コンパイル (プログラムチェック)]、または [PLC] メニューから [全プログラムコンパイル (チェック)])

例) FB インスタンスエリアの非保持領域として、W0~511 を指定していて、ラダープログラム上で、例えば W0.00 を使用していると、コンパイル時に「エラー：… (略) …アドレス-W0.00 はファンクションブロック用に予約されています。」と表示されます。



参考

変数の FB インスタンスエリアへの割当は、変数の追加または削除を行った時点で、自動的に再割り当てが行われます。

ただし、1つのインスタンスは連続したアドレスが必要となるため、もし再割当てで連続したアドレスが確保できないと、すべての変数に別のアドレスを割り当て直します。その結果、未使用領域ができてしまいます。

このような場合、未使用領域をなくし、割り当て領域を効率的に使用するために、最適化操作を実行することができます。

■ コメント

日本語も使用可能であり、最大全角 15000 文字まで。

■ 複数のインスタンス化

● 同一インスタンスの呼出

複数箇所からひとつのインスタンスを呼び出すことができます。この場合、内部変数は共通です。

● インスタンスの複数化

ひとつのファンクションブロック定義に対して複数のインスタンスを生成することができます。

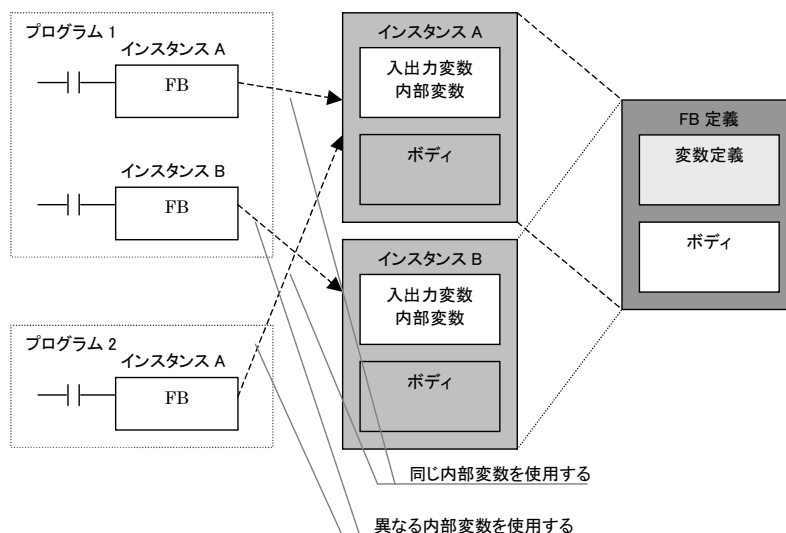
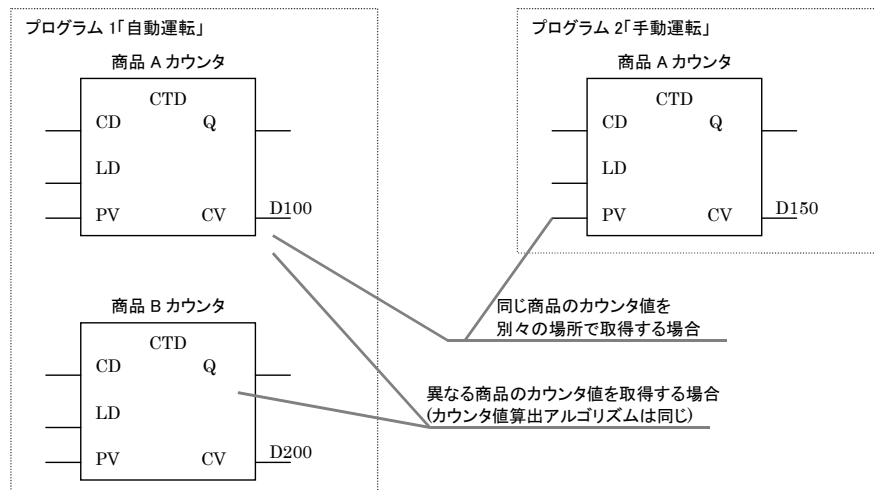
この場合、内部変数は各々のインスタンスで値が異なります。

例) 商品 A と商品 B をカウントします。

ファンクションブロック定義「ダウンカウンタ (CTD)」を用意し、商品 A に対するカウンタ・商品 B に対するカウンタを設ける。

プログラムは「自動運転」用と「手動運転」用に分けられており、ユーザは適宜切り替えを行います。

そのような場合、1 つのファンクションブロック定義から複数のインスタンスを生成し、複数の箇所から同じインスタンスを参照できる必要があります。



2-3 インスタンスの仕様

2-3-2 パラメータの仕様

2-3-2 パラメータの仕様

ユーザが、入力パラメータ、出力パラメータに設定できるデータは以下のとおりです。

項目	設定可能なデータ
入力パラメータ	値*1、アドレス、プログラム側の変数（グローバル変数、ローカル変数）*2 注1：パラメータから入力変数へ渡されるデータは、入力変数のデータサイズ分の値自体です（パラメータにアドレスを設定しても、アドレスは渡りません）。 注2：入力パラメータは、設定必須です。1つでも設定していない入力パラメータがあると、致命的なエラーとなり、実機 PLC へ転送できません。
出力パラメータ	アドレス、プログラム側の変数（グローバル変数、ローカル変数）*2
入出力パラメータ	アドレス、プログラム側の変数（グローバル変数、ローカル変数）

*1：パラメータへの、値の入力方法一覧

入力変数のデータ型	内容	ビットサイズ	パラメータへの値の入力方法	値の最大範囲
BOOL	ブール型	1	P_Off、P_On	0 (FALSE)、1 (TRUE)
INT	整数型	16	+の値:&または+の後に整数 -の値:-の後に整数	-32768 ~ +32767
DINT	倍精度整数型	32		-2147483648 ~ +2147483647
LINT	4 倍精度整数型	64		-9223372036854775808 ~ +9223372036854775807
UINT	符号なし整数型	16	+の値:&または+の後に整数	&0 ~ 65535
UDINT	符号なし倍精度整数型	32		&0 ~ 4294967295
ULINT	符号なし 4 倍精度整数型	64		&0 ~ 18446744073709551615
REAL	実数型	32	+の値:+の後に実数（小数点あり） -の値:-の後に実数（小数点あり）	-3.402823×10 ³⁸ ~-1.175494×10 ⁻³⁸ 、0、+1.175494×10 ⁻³⁸ ~+3.402823×10 ³⁸
LREAL	倍精度実数型	64		-1.79769313486232×10 ³⁰⁸ ~-2.22507385850720×10 ⁻³⁰⁸ 、0、+2.22507385850720×10 ⁻³⁰⁸ ~+1.79769313486232×10 ³⁰⁸
WORD	長さ 16 のビット列型	16	#の後に、16 進数（4 桁以下） または &または+の後に 10 進数	#0000~FFFF または&0 ~ 65535
DWORD	長さ 32 のビット列型	32	#の後に、16 進数（8 桁以下） または &または+の後に 10 進数	#00000000~FFFFFFFF または&0 ~ 4294967295
LWORD	長さ 64 のビット列型	64	#の後に、16 進数（16 桁以下） または &または+の後に 10 進数	#0000000000000000~FFFFFFFFFFFFFFFF または&0 ~ 18446744073709551615

*2：ファンクションブロックの入力変数または出力変数のデータサイズと、プログラム側の変数（グローバル変数、ローカル変数）のデータサイズが、一致していることが必要です（下表参照）。

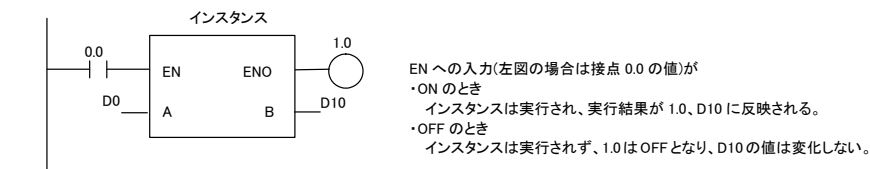
ビットサイズ	ファンクションブロックの変数のデータ型	プログラム側の変数（グローバル変数、ローカル変数）のデータ型
1	BOOL	BOOL
16	INT、UINT、WORD	INT、UINT、UINT BCD、WORD
32	DINT、UDINT、REAL、DWORD	DINT、UDINT、UDINT BCD、REAL、DWORD
64	LINT、ULINT、LREAL、LWORD	LINT、ULINT、ULINT BCD、LREAL、LWORD
1 以外	BOOL 以外	CHANNEL、NUMBER (*1)

*1：プログラム側の変数の「NUMBER」は、入力パラメータにのみ設定可能。また、入力可能な値は、ファンクションブロックの変数のデータ型のサイズの範囲内のみ可能。

2-3-3 動作仕様

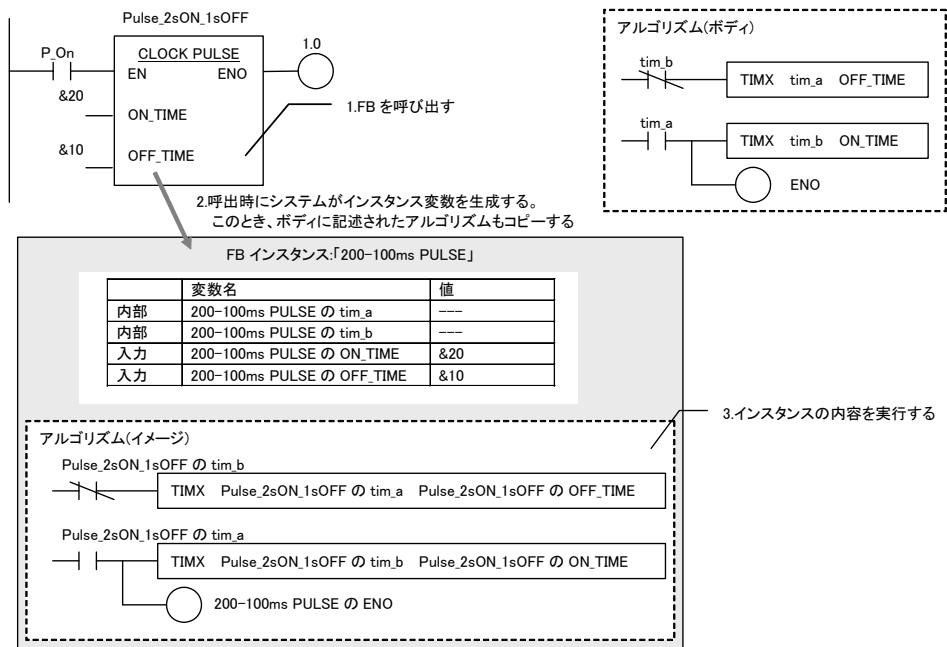
■ インスタンスの呼出

ユーザは任意の場所でインスタンスを呼び出すことができます。
インスタンスは EN への入力が入力 ON のとき、実行されます。

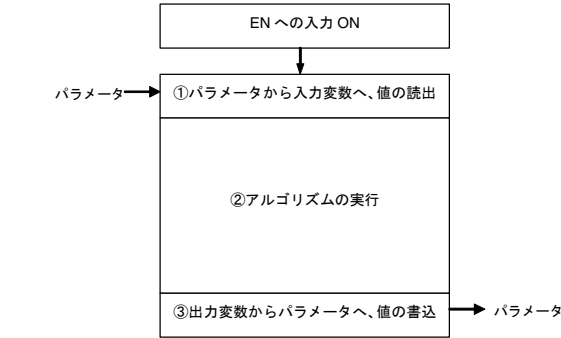


■ 実行時の動作詳細

ファンクションブロックの入力変数 EN が ON の場合、システムはファンクションブロックを呼び出します。
呼出時にインスタンス変数を生成し、ファンクションブロック定義に記述されたアルゴリズムをコピーします。
ファンクションブロックのインスタンスを実行します。



実行順は、以下のように、①パラメータから入力変数への読み出し、→②アルゴリズムの実行→③出力変数からパラメータへの書き込みの順です。

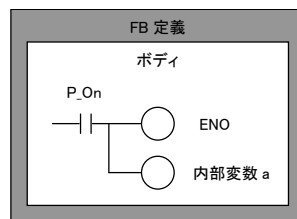
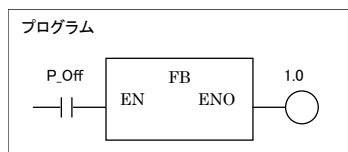


したがって、アルゴリズムの中で、パラメータの受け渡しは実行されません。
また、アルゴリズムの実行によって出力変数が変更されなかった場合、出力パラメータの値は、前回の値が保持されたままとなります。

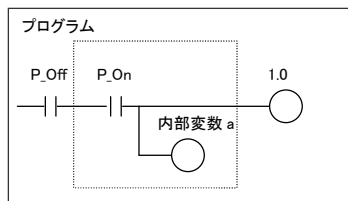
■ 非実行時の動作

EN が OFF のときは、ファンクションブロックが呼び出されないため、ファンクションブロック定義の内部変数の値は変わりません（保持されます）。

また同様に、EN が OFF のときは、出力変数の値も変わりません（保持されます）。



実行結果:
出力変数 1.0 は OFF となるが、
内部変数 a は、前回の値を保持したままとなる。



仮にファンクションブロック定義ではなく、
プログラムに直接記述されている場合は、
両接点 1.0、a とも OFF となる。

お願い

インスタンスは、EN が OFF の間は実行されません。したがって、EN が OFF となっても、微分命令/タイマ命令の初期化は、実行されません。

そのため、ファンクションブロック内で微分命令/タイマ命令を使用する時は、EN の入力条件を、常時 ON フラグ（P_On）に設定してください。

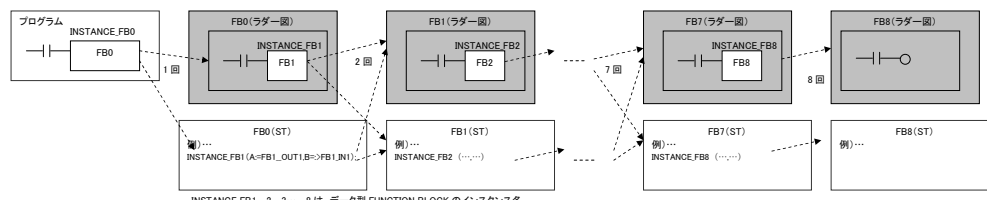
また、EN の入力条件に常時 ON フラグを設定している場合でも、インターロック命令/ジャンプ命令/サブルーチン命令によって常時 ON フラグが無効とならないようにしてください。

■ ネスティング

CX-Programmer Ver.6.0 以降では、ファンクションブロックから他のファンクションブロックを呼び出すことが可能です。

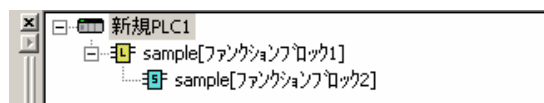
最大ネスティング数は 8 段（プログラムからファンクションブロックを呼び出すことを含む）です。

呼び出し元ファンクションブロックと呼び出し先ファンクションブロック内の言語の組み合わせは、ラダー言語/ST 言語のいずれでも可能です。



注) ネスティング元とネスティング先の言語の組み合わせは、ラダー言語、ST 言語の組み合わせすべて任意

ファンクションブロックのネスティングの階層関係をつリー表示することもできます（FB インスタンスビューアによる）。



なお、ネスティングしているファンクションブロック定義は、呼び出している（ネストしている）ファンクションブロック定義を含めて、1 つのファンクションブロックライブラリファイル（.xcf）に保存されます。

2-4 制限事項

2-4-1 ラダー言語使用時の命令制限

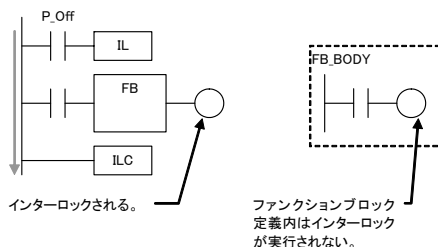
2

仕様

- ファンクションブロック定義内で使用不可の命令
CS/CJ/NSJ シリーズ コマンドリファレンスマニュアル (No.SBCA-351) を参照してください。
- AT 指定できない I/O メモリ
以下の I/O メモリは、AT 指定できません。
 - ・インデックスレジスタ（間接指定、直接指定とも）／データレジスタ（注：AT 指定でなく、直接アドレスを入力できます）
 - ・DM/EM 間接指定（@：BIN モード、*：BCD モードとも）
- 命令のオペランドにアドレスを直接入力できる I/O メモリ
 - ・インデックスレジスタ（間接指定、直接指定とも）／データレジスタは、変数でなく、アドレスを直接入力できます。
直接指定時：「IR0～15」、間接指定時：「,IR0～15」、定数オフセット指定時：例）「+5,IR0」、DR オフセット指定時：「DR0,IR0」、オートインクリメント指定時：「,IR0++」、オートデクリメント指定時：「--,IR0」を、命令のオペランドに入力可能。
 - ・それ以外の I/O メモリは、命令のオペランドにアドレスを直接入力できません。
- 入力変数、出力変数、または入出力変数との受け渡し用のパラメータとしては使用できない I/O メモリ
以下の I/O メモリは、入力変数、出力変数、または入出力変数との受け渡し用のパラメータとして使用できません。
 - ・インデックスレジスタ（間接指定、直接指定とも）／データレジスタ
 - ・DM/EM 間接指定（@：BIN モード、*：BCD モードとも）

● インターロックの使用制限

インターロック領域内にファンクションブロックの呼出がある場合、ファンクションブロック定義内は実行されません。
サブルーチンと同様の動作となります。



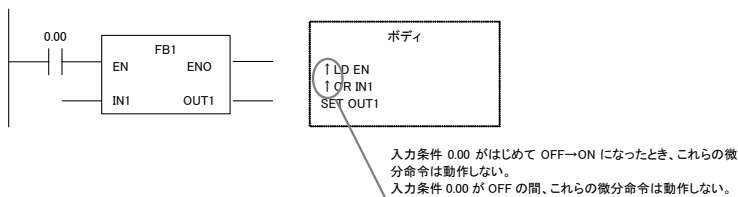
● ファンクションブロック定義内での微分命令の使用制限

インスタンスは、EN が OFF の間は実行されません。そのため、ファンクションブロック定義内で微分命令*1を使用する場合、以下の注意が必要です。

*1：DIFU 命令/DIFD 命令、動作オプション@付き/%付き命令

- ・EN が OFF の間は、前回値（EN が ON のときの最終値）を保持したままとなり、微分命令は動作しません。
- ・EN が ON となって初回の条件成立時は、前回値は前に EN が ON であったときの最終値との差がチェックされるため、微分命令は正しく動作しません。（EN が ON の状態で、次の条件成立時は、微分命令は正しく動作します。）

例)



したがって、ファンクションブロック定義内で微分命令を使用する場合、EN の入力条件は、常時 ON フラグ（P_On）として、ファンクションブロック定義内で、入力条件を組んでください。

例)



● 以下の命令の第1オペランドを指定するとき、10進の数値の前に#を付けてください（&は使用できません）。

MILH、MILR、MILC、DIM、MSKS、MSKR、CLI、FAL、FALS、TKON、TKOF

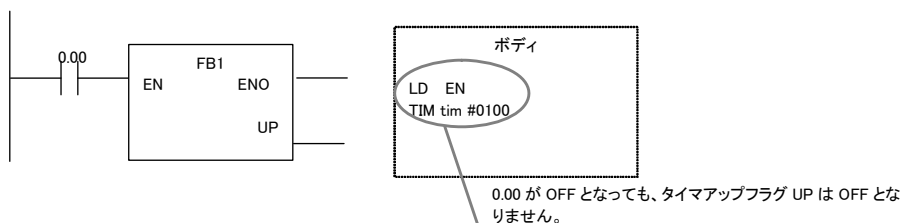
● CNR、CNRX（タイマ／カウンタリセット）命令は、ファンクションブロック内では、複数のタイマ／カウンタに対して一括リセットすることはできません。

（第1オペランド（タイマ／カウンタ番号1）と第2オペランド（タイマ／カウンタ番号2）に異なる変数を指定することはできず、必ず、同じ変数を指定してください。）

● ファンクションブロック定義内でのタイマ命令の使用制限

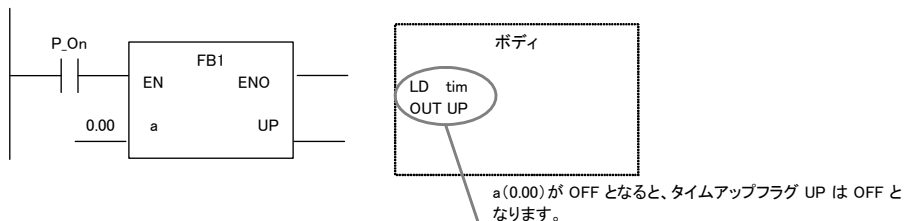
- ・ インスタンスは、EN が OFF のときは実行されません。そのため、ファンクションブロック定義内でタイマ命令を使用する場合、以下の注意が必要です。
EN が OFF となっても、タイマ命令は初期化されません。したがって、EN が ON で、タイマが起動した後、EN が OFF となっても、タイムアップフラグは OFF とはなりません。

例)



したがって、ファンクションブロック定義内でタイマ命令を使用する場合、EN の入力条件は、常時 ON フラグ (P_On) として、ファンクションブロック定義内で、入力条件を組んでください。

例)



- ・ タイマを使用した同じインスタンスを複数箇所にて同時に使用すると、タイマ二重使用となります。

2-4-2 ST 言語時の命令制限

■ ファンクションブロック内で ST 言語を使用する時の制限

- ・使用可能なものは下記に限定されます。
 - ・代入文
 - ・選択文 (CASE 文、IF 文)
 - ・繰り返し文 (FOR 文、WHILE 文、REPEAT 文、EXIT 文)
 - ・RETURN 文
 - ・ファンクションブロック呼び出し文
 - ・四則演算
 - ・論理演算
 - ・比較演算
 - ・数値演算関数
 - ・算術演算関数
 - ・標準文字列関数
 - ・数値文字列変更関数
 - ・オムロン拡張関数
 - ・コメント

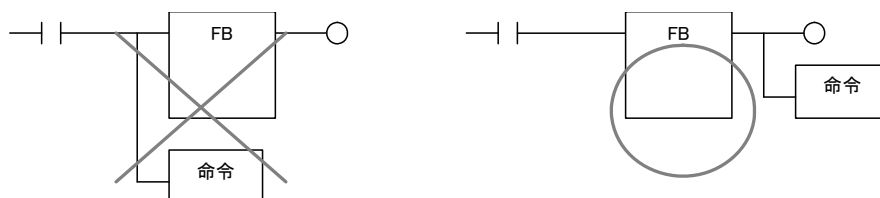
その他の詳細については、ストラクチャードテキスト編 第5章「ST 言語の仕様」を参照してください。

2-4-3 ラダー/ST 言語の共通の制限

■ 回路上のインスタンスの配置制限

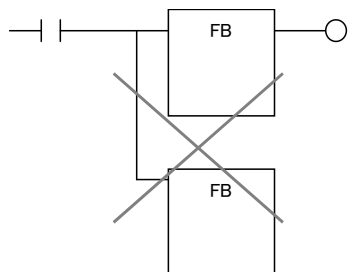
● インスタンスの左側での回路の分岐不可

インスタンスの左側で回路を分岐することができません。
右側は可能です。



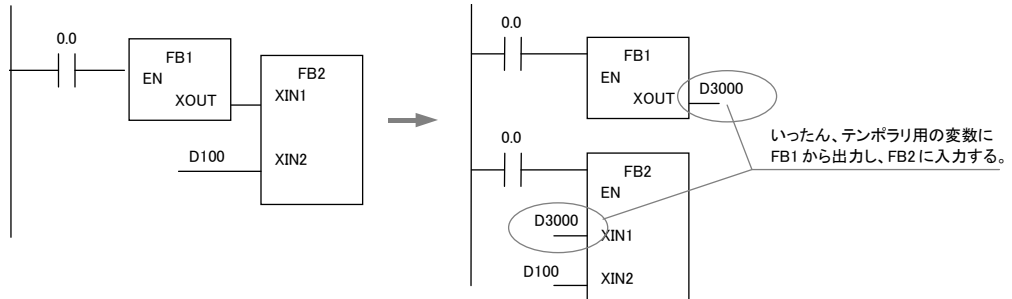
● ひとつの回路には 1 つのインスタンスのみ

ひとつの回路に複数のインスタンスを持つことができません。



● ファンクションブロックの連結不可

ファンクションブロックの出力をそのまま別のファンクションブロックの入力とすることはできません。
この場合、出力と入力の間にテンポラリとして変数をユーザが用意する必要があります。



■ タスク単位でのダウンロードの制限

ファンクションブロックを含むタスクを、タスク単位でダウンロードすることはできません（アップロードは可能です）。

■ プログラミングコンソールで読み出した時のインスタンスの表示制限

CX-Programmer で作成したユーザプログラムを CPU ユニットにダウンロードして、プログラミングコンソールで読み出すと、インスタンスはすべて、「？」と表示されます（インスタンス名は表示されません）。

■ オンラインエディットの制限事項

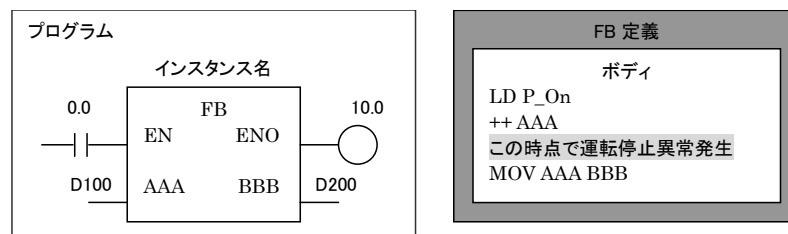
オンラインエディットによって、CPU ユニット内のユーザプログラムに対して、以下の操作は、実行できません。

- ・ ファンクションブロック定義（変数テーブル、アルゴリズム）の削除
- ・ インスタンスの挿入、インスタンス名変更

注：インスタンスの受け渡し用パラメータの変更、インスタンス以外の命令変更、インスタンスの削除は可能です。

■ 異常関連

ファンクションブロック定義実行中に CPU ユニットにて運転停止異常が発生した場合は、発生した時点でラダー実行を停止します。



上記の場合、「MOV AAA BBB」は実行されず、出力変数 D200 はファンクションブロック実行前の値が保持されます。

■ FB インスタンスエリアへのアクセス禁止

ファンクションブロックを実現する際、システムはインスタンスの内部変数や入力変数、出力変数、または入出力変数を格納するエリアを必要とします。

・ CJ2 シリーズ CPU ユニットの場合

FB インスタンスエリア	開始アドレス初期値	サイズ初期値	割付可能エリア
非保持領域	H512	896	CIO、WR、HR、DM、EM ^{*1}
保持領域	H1408	128	HR、DM、EM ^{*1}
タイマ領域	T3072	1024	TIM
カウンタ領域	C3072	1024	CNT

*1. 以下の EM バンクを指定した場合、強制セット／リセットが可能となります。

形 CJ2H-CPU64/65(-EIP)	EM バンク 3
形 CJ2H-CPU66(-EIP)	EM バンク 6～9
形 CJ2H-CPU67(-EIP)	EM バンク 7～E
形 CJ2H-CPU68(-EIP)	EM バンク 11～18

・ CS/CJ シリーズ CPU ユニット Ver.3.0 以降、NSJ の場合

FB インスタンスエリア	開始アドレス初期値	サイズ初期値	割付可能エリア
非保持領域	H512	896	CIO WR HR DM EM
保持領域	H1408	128	HR DM EM
タイマ領域	T3072	1024	TIM
カウンタ領域	C3072	1024	CNT

・ FQM1 の場合

FB インスタンスエリア	開始アドレス初期値	サイズ初期値	割付可能エリア
非保持領域	5000	1000	CIO、WR、DM
保持領域	なし		
タイマ領域	T206	50	TIM
カウンタ領域	C206	50	CNT

・ CP1 シリーズ CPU ユニットの場合

FB インスタンスエリア	開始アドレス初期値	サイズ初期値	割付可能エリア
非保持領域	H512	896	CIO、WR、HR、DM ^{*1}
保持領域	H1408	128	HR、DM ^{*1}
タイマ領域	T3072	1024	TIM
カウンタ領域	C3072	1024	CNT

*1. CP1L-L の DM エリアは以下のようにになっています。

アドレス	CP1L-L
D0000～D9999	あり
D10000～D31999	存在しない
D32000～D32767	あり

・ CP2E シリーズ CPU ユニットの場合

FB インスタンスエリア	開始アドレス初期値	サイズ初期値	割付可能エリア
非保持領域	H512	896	CIO、WR、HR、DM
保持領域	H1408	128	HR、DM
タイマ領域	T256	256	TIM
カウンタ領域	C256	256	CNT

ユーザプログラム上で「FB インスタンスエリア」にアクセスする命令が存在すると、CX-Programmer は、以下のときにエラーを出力します。

- ・ユーザによるプログラチェック実行時（[プログラム] メニューから [コンパイル（プログラムチェック）]、または [PLC] メニューから [全プログラムコンパイル（チェック）]）
- ・オンラインエディットをして書き込みをしようとしたとき（書き込みは不可）

■ ネスティング時に構造体変数をパラメータとして指定する際の制限

ファンクションブロックの中で別のファンクションブロックを呼び出すとき（ネスティング時）に、呼び出すファンクションブロックのパラメータとして、構造体変数の特定のメンバのみを指定することはできません。構造体変数全体を指定してください。

■ アドレス加算コピー機能の制限

選択したセクション内でファンクションブロックが使用されている場合は、アドレス加算コピー機能は使用できません。

2-5 ファンクションブロック使用のガイドライン

以下に、CX-Programmer によってファンクションブロックを使用するときの、ガイドラインを示します。

2-5-1 変数のデータ型の決め方

● 1、2 または 4CH 構成で、ひとつの数値データ：整数型を使用

1、2 または 4CH 構成で、ひとつの数値データとして扱うものは、以下のデータ型を使用します。

- ・ INT、UINT
- ・ DINT、UDINT
- ・ LINT、ULINT

注：いずれも、範囲内に収まるのであれば、符号付きを使用する

● 1、2 または 4CH 構成で、ひとつかたまりのデータ：ワードデータ型を使用

1、2 または 4CH 構成で、ひとつかたまりのデータ(数値としての意味を持たないもの)として扱うものは、以下のデータ型を使用します。

- ・ WORD
- ・ DWORD
- ・ LWORD

● 文字列データ：文字列型を使用

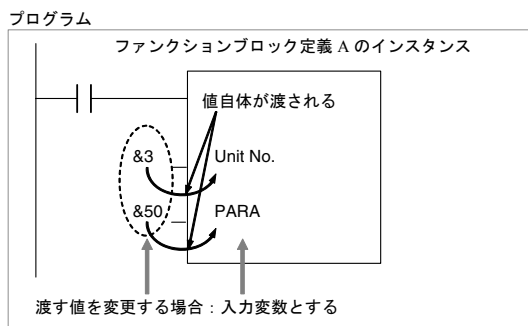
文字列として扱うものは、以下のデータ型を使用します。

- ・ STRING

2-5-2 変数種類（どれを入力変数、出力変数、入出力変数、外部変数、内部変数とするか）の決め方

● 渡す値を変更したい変数は、入力変数

プログラム上に貼り付けて、インスタンスごとに、ファンクションブロックに渡す値（アドレス自体ではありません）を変更したい場合、それを「入力変数」とします。

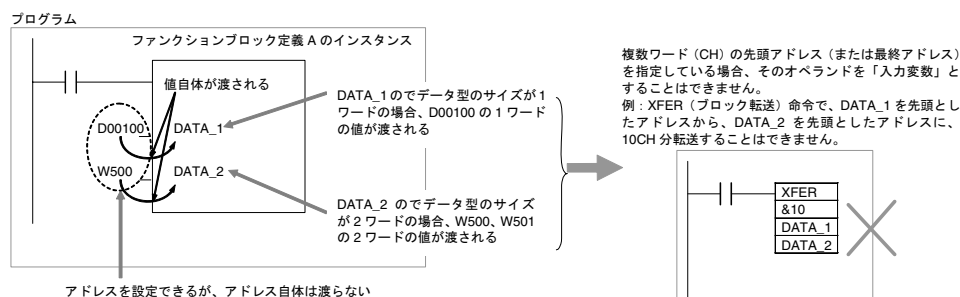


2-5 ファンクションブロック使用のガイドライン

2-5-2 変数種類（どれを入力変数、出力変数、入出力変数、外部変数、内部変数とするか）の決め方

ただし、以下の2点にご注意ください。

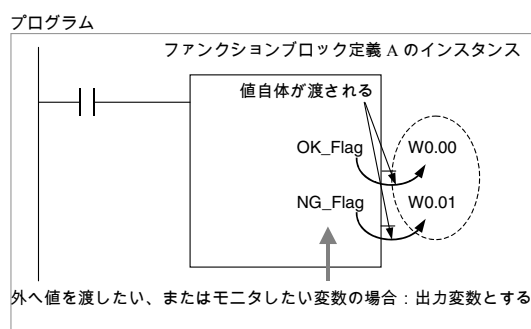
- ・入力パラメータにアドレスを設定することは可能ですが、その場合、入力変数にアドレス自体を渡すことはできません（入力パラメータにアドレスを設定しても、入力変数のデータ型サイズ分の値が、ファンクションブロックに渡されます）。
したがって、ファンクションブロック内の命令のオペランドが、複数ワード（CH）の先頭アドレス（または最終アドレス）を指定している場合、そのオペランドを「入力変数」とすることはできません。
その場合、「内部変数」としてAT指定するか、または（後述の「2-4-4 内部変数の配列型指定」で示すように）「配列型指定」の「入出力変数」（入力パラメータに先頭アドレスを設定）として配列型変数の先頭（または最終）を指定するか（CX-Programmer Ver.7.0 以降のみ）、「配列型指定」の「内部変数」として配列型変数の先頭（または最終）を指定するか、または外部変数を指定してください。



- ・入力パラメータから「入力変数」には、（アルゴリズム内命令実行のタイミングでなく）アルゴリズムの前に一括で、値が渡されます。
したがって、ファンクションブロックのアルゴリズム内の命令実行のタイミングで、パラメータから「入力変数」に値を渡すことが必要な場合は、「入力変数」としないで、「内部変数」または「外部変数」としてください。

● 外へ値を渡したい、またはモニタしたい変数は、出力変数

プログラム上に貼り付けて、インスタンスごとに、ファンクションブロックから外（プログラム）へ値を渡したい場合、または値をモニタしたい場合、それを「出力変数」とします。



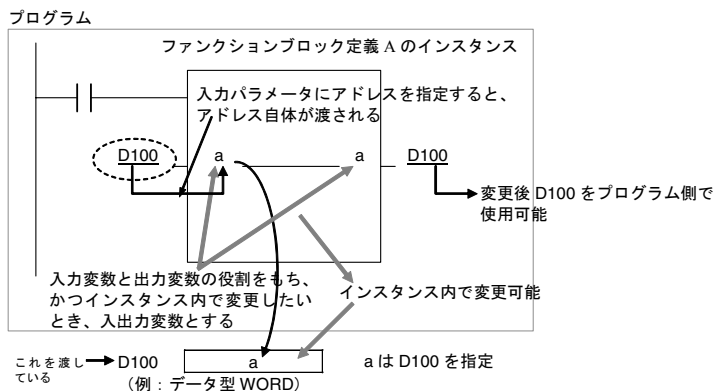
ただし、以下の点にご注意ください。

- ・「出力変数」から出力パラメータには、（アルゴリズム内命令実行のタイミングでなく）アルゴリズムの後に一括で、値が渡されます。
したがって、ファンクションブロックのアルゴリズム内の命令実行のタイミングで、「出力変数」から出力パラメータに値を渡すことが必要な場合は、「出力変数」としないで、「内部変数」または「外部変数」としてください。

2-5 ファンクションブロック使用のガイドライン

2-5-2 変数種類（どれを入力変数、出力変数、入出力変数、外部変数、内部変数とするか）の決め方

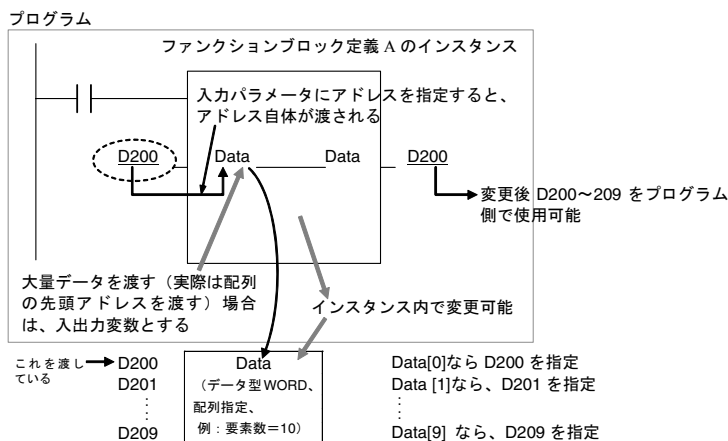
- 入力パラメータとして外部から渡された値をファンクションブロック内で変更し、その結果を出力パラメータとして外部に戻すための変数は、**入出力変数**
プログラム上に貼り付けて、ファンクションブロックに対して入力パラメータ/出力パラメータ双方の役割を持たせたい場合、それを「入出力変数」とします。
内部の動作として、入出力変数にはパラメータのアドレスが渡されますが、ファンクションブロック内で入出力変数を使うときには、他の種類の変数と同じように使用することができます。



- 大量のデータを受け渡ししたい変数は、**配列指定した入出力変数**

入出力変数は、配列指定が可能です（入力変数、出力変数は、配列指定できません）。したがって、入出力変数が配列指定されている場合には、パラメータのアドレスを先頭アドレスとしたデータサイズ分のアドレス範囲を、ファンクションブロック内で使用することができます。

そのため、ファンクションブロック外部から大量データを渡したいときには、入出力変数を使用できます。



- コンディションフラグ・クロックパルス・特殊補助リレー、プログラム上で使用しているグローバル変数は、**外部変数**
コンディションフラグ（常時 ON フラグ、=フラグなど）、クロックパルス（1 秒クロックなど）、あらかじめ登録されている特殊補助リレー（運転開始時 1 サイクル ON フラグなど）、プログラム上で使用しているグローバル変数は、システム定義の「外部変数」とします。

- それ以外の内部割当する変数および AT 指定する必要がある変数は、内部変数
上記以外の、1) アドレスを内部割当する変数、2) アドレスを「AT 指定」する必要がある変数（I/O 割付アドレス、高機能ユニット割付アドレスが特定化されている場合など）は、「内部変数」とします。
- 3) 「配列指定」をする必要がある変数は、「入出力変数」または「内部変数」とします。
「AT 指定」、「配列指定」が必要な条件については、各々、後述の 2-4-3 項「内部変数の AT 指定」、2-4-4 項「入力変数または内部変数の配列型指定」を参照してください。

2-5-3 内部変数の AT 指定

以下の場合、必ず、内部変数として、AT 指定を使用してください。

- ・ 基本 I/O ユニットまたは高機能ユニットの割付アドレスで、かつその割付アドレスをグローバル変数へ登録してその変数を外部変数として指定することができない場合（グローバル変数とのデータセットが保証できないときなど）
注：高機能ユニットの割付アドレスの場合、後述のインデックスレジスタを使用した指定方法では、割付エリア先頭アドレスを AT 指定することが必要です（2-4-5 項「高機能ユニットの割付アドレスの指定方法」参照）。
- ・ あらかじめ外部変数に登録されている以外の特殊補助リレーで、かつその特殊補助リレーをグローバル変数へ登録してその変数を外部変数として指定しない場合
- ・ ネットワーク上の他ノードの、PLC のアドレス指定時：例）SEND 命令の送信先（相手ノード）受信開始 CH 番号、RECV 命令の送信要求元（自ノード）受信開始 CH 番号
- ・ 命令のオペランドが、複数ワード（CH）の先頭アドレス（または最終アドレス）を指定している場合で、かつそのオペランドを配列指定することができない場合（例：配列要素数を特定化できないとき）

2-5-4 入力変数または内部変数の配列型指定

■ 命令のオペランドが、複数ワード (CH) の先頭アドレス (または最終アドレス) を指定している場合：配列型を使用

命令のオペランドが、複数ワード (CH) の先頭アドレス (または最終アドレス) を指定している場合*1、命令は、AT 指定または内部割当て後の「アドレス」にしたがって動作します (したがって、変数のデータ型および配列の要素数は、命令動作とは無関係となります)。

必ず AT 指定の変数か、または命令が処理するデータサイズと一致した要素数の配列型の変数を指定してください。

*1: 例えば、ブロック転送 (XFER) 命令の転送元先頭アドレス/転送先先頭アドレス、SEND 命令の送信元送信開始 CH 番号、応用命令のコントロールデータなど。

詳細は、2-5 項「オペランドが複数ワードの先頭/最終アドレスを指定している命令の注意事項」を参照してください。

入出力変数を使用する場合は、複数ワード (CH) の先頭アドレスを入力パラメータとして設定してください。

内部変数を使用する場合は、以下の方法で配列指定をします。

1. 必要サイズの要素数の、配列指定の内部変数を用意します。

注：このとき、必ず、命令が処理をするサイズと同一の要素数を、確保してください。

各命令が処理をするサイズについては、2-6 項「命令の使用可否と、オペランドごとの使用変数の制限一覧」の「AT 指定または配列指定をすることが必要 (カッコ内は、変数の必要データワードサイズ)」の列を参照してください。

2. ファンクションブロック定義内で、MOV 命令などで、各配列要素にデータをセットします。

3. 配列型変数の先頭 (または最終) をオペランドに指定します。

これにより、複数ワード (CH) の先頭アドレス (または最終アドレス) を指定することができます。

以下のケースがあります。

・ 複数 CH でひとつのデータを意味するもの

例) FREAD 命令の「ディレクトリおよびファイル名」

・ 変数テーブル：

入出力変数または内部変数、データ型：WORD 型、配列指定・要素数 10、変数名：file_name[0]~file_name[9]

・ データのセットと、ファンクションブロック内の演算

－入出力変数の場合：

入力パラメータに、データの先頭アドレス (例：D100) を設定します。プログラム側で、データ (#5C31、#3233、#0000、...) を、D100~109 にあらかじめセットしておきます。

FREAD (略) (略) read_num[0] (略) ← 配列変数の先頭をオペランドに指定

－内部変数の場合：

ファンクションブロック内のラダー言語で、各配列にデータをセットします。

```

MOV #5C31 file_name[0]
MOV #3233 file_name[1]
MOV #0000 file_name[2]
      !
FREAD (略) (略) read_num[0] (略) ← 配列変数の先頭をオペランドに指定

```

各配列にデータをセット

・複数 CH でコントロールデータ的な役割を果たすもの

例) FREAD 命令の「読出 CH 数・読出開始位置」

・変数テーブル：

入出力変数または内部変数、データ型：DINT 型、配列指定・要素数 3、変数名：
read_num[0]～read_num[2]

・データのセットと、ファンクションブロック内の演算

－入出力変数の場合：

入力パラメータに、データの先頭アドレス（例：D200）を設定します。プログラム側で、データを、D200～105 にあらかじめセットしておきます。

FREAD (略)file_name[0] (略) (略) ← 配列変数の先頭をオペランドに指定

－内部変数の場合：

ファンクションブロック内のラダー言語で、各配列にデータをセットします。

・ラダー言語：

```

MOVL &100 read_num[0] (読出 CH 数)
MOVL &0 read_num[1] (開始位置)
      } ← 各配列にデータをセット
FREAD (略)file_name[0] (略) (略) ← 配列変数の先頭をオペランドに指定

```

・複数 CH でひとつのデータ群と見なすもの

あらかじめ許容可能なデータサイズを決め、最大データサイズ分の配列を確保する必要があります。

例) FREAD 命令の「読出データ」

・変数テーブル：

入出力変数または内部変数、データ型：WORD 型、配列指定・要素数 100、変数名：read_data[0]～read_data[99]

・データのセットと、ファンクションブロック内の演算

－入出力変数の場合：

入力パラメータに、読出データの格納先頭アドレス（例：D200）を設定します。

FREAD (略) (略) (略) read_data[0]

－内部変数の場合：

FREAD (略) (略) (略) read_data[0]

■ 除算（ラダー言語の場合のみ）：INT 型の配列型を使用

ラダー言語の場合、除算（*I*）命令の演算結果には INT 型の配列を用います。

除算（*I*）命令の出力は、D:商、D+1:余りとなります。

このように、ラダー言語の場合は、余りを算出することができます。

注：ST 言語の場合、除算の出力に配列を用いる必要はありません。なお、ST 言語で余りを直接算出することはできません。余り = 被除算値 - (除算値 * 商) で求める必要があります。

2-5-5 高機能ユニットの割付アドレスの指定方法 (号機 No./ユニット番号から自動算出)

ファンクションブロック定義内で、入力パラメータとして、号機 No./ユニット番号を渡し、その値をもとに高機能ユニットの割付アドレスを特定したいときは、以下のように、インデックスレジスタ (IR0~15) (の定数オフセット間接指定) を使用します。

注：ファンクションブロック内でのインデックスレジスタの使用方法については、後述の 2-4-6 項「インデックスレジスタの使用方法」を参照してください。

例：割付リレーエリアをファンクションブロック内で特定化する場合 (割付 DM エリアも同様)

例) 高機能 I/O ユニットの場合：

- ・変数：号機 No. を入力変数とし、かつ割付先頭アドレスを内部変数 (データ型 WORD) として AT 指定で 2000CH を指定する。

- ・プログラム：

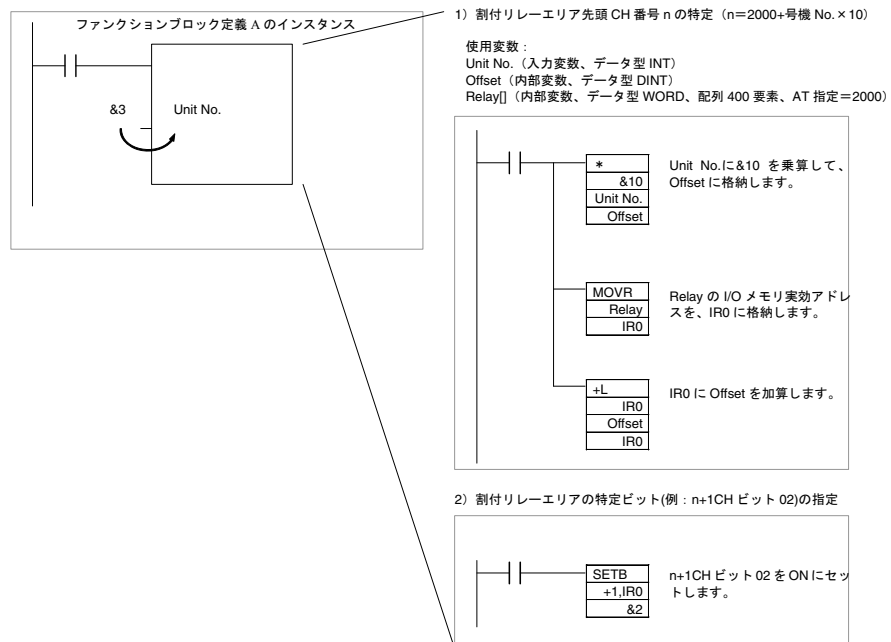
1. 「号機 No.」 (入力変数) に &10 を乗算し、号機 No. 分の「オフセット」 (内部変数、データ型 DINT) を作成します。
2. MOVR (インデックスレジスタ設定) 命令で、「割付先頭アドレス」 (内部変数、AT=2000CH) の I/O メモリ実効アドレスをインデックスレジスタ (例：IR0) に格納します。
3. インデックスレジスタ (例：IR0) 内の I/O メモリ実効アドレスに号機 No. 分の「オフセット」を加算します。

2) 割付リレーエリアの特定ビット (例：n+aCH ビット b) を、指定します。

- ・プログラム：

- ・CH アドレス：インデックスレジスタの定数オフセットの間接指定 (例：+a,IR0) で指定します。
- ・ビットアドレス：CH 内ビットアドレスを指定できる命令 (例：書込時：SETB 命令の第 2 オペランドに &b、読み出し時：TST 命令の第 2 オペランドに &b) で指定します。

例) 高機能 I/O ユニットの場合

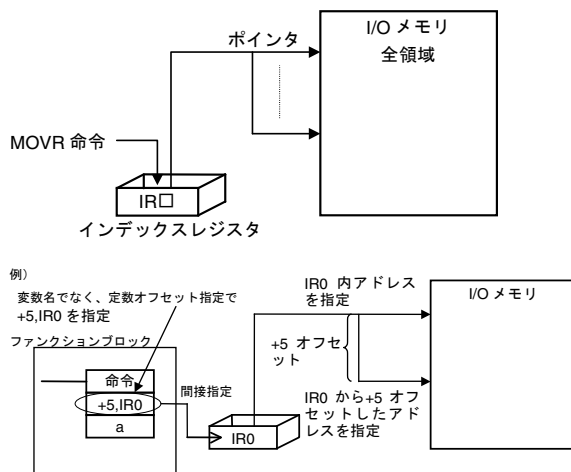


2-5-6 インデックスレジスタの使用法

インデックスレジスタ（IR0～15）は、I/O メモリのアドレスを指定するためのポインタとして機能します。

ファンクションブロック内では、このインデックスレジスタを、変数名でなく IR0～15 でアドレスを直接指定できます（インデックスレジスタ直接指定時：IR0～15、インデックスレジスタ間接指定時：,IR0～15）。

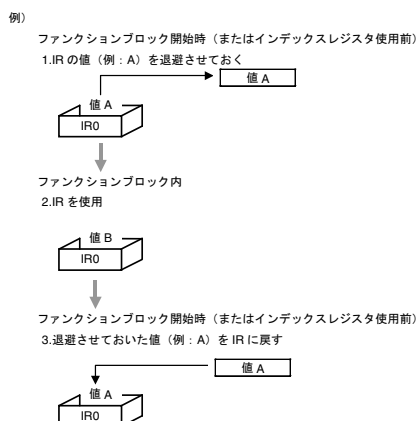
注：MOVR 命令によって、インデックスレジスタに I/O メモリ実効アドレス格納した上で、一般の命令でインデックスレジスタを間接指定します。これにより、I/O メモリ全領域を可変的に指定することができます。



お願い

- ・ファンクションブロック内でインデックスレジスタ（IR0～15）を使用する場合、他のファンクションブロック内またはプログラム上で、同一のインデックスレジスタを使用していると、互いに競合してしまい、プログラムが正しく動作しません。

そのため、インデックスレジスタ（IR0～15）を使用する場合、必ず、ファンクションブロック開始時（またはインデックスレジスタ使用前）にその時点のインデックスレジスタの値をどこかに退避させておき、ファンクション終了時（またはインデックスレジスタ使用后）に、使用したインデックスレジスタの値を、退避させておいた値に戻すように、プログラムを組んでください。

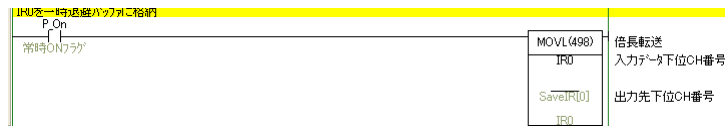


- ・インデックスレジスタを使用する場合、必ず、値を設定してから使用してください。値が設定されていない状態で使用した場合、動作は不定となります。

■ 使用例

ファンクションブロック内でインデックスレジスタ（IR0～15）を使用する例を以下に示します。

●インデックスレジスタ使用前に、インデックスレジスタの値を退避させておく

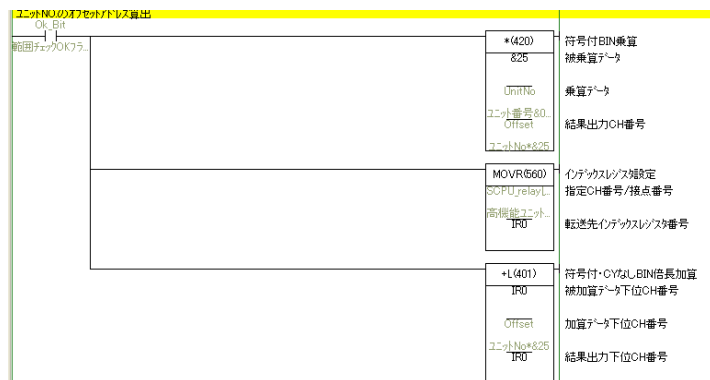


本ファンクションブロック内でインデックスレジスタを使用する場合、本ファンクションブロック終了時（またはインデックスレジスタ使用后）にインデックスレジスタの値を元に戻すため、本ファンクション開始時（またはインデックスレジスタ使用前）に、インデックスレジスタの値の退避処理をしておきます。

例）インデックスレジスタ IR0 の内容を、SaveIR[0]（内部変数、データ型 DINT、配列 1 要素）に格納し、退避させておきます。

●インデックスレジスタ使用例

- 1) インデックスレジスタに値を設定
（割付リレーエリア先頭 CH 番号 n の I/O メモリ実効アドレスを格納）



例）ファンクションブロックに渡された CPU 高機能ユニットのユニット番号（&0～15）をもとに、CPU 高機能ユニット割付リレーエリアの割付先頭「1500CH+ユニット番号×25」CH の I/O メモリ実効アドレスを IR0 に格納します。

手順：

すでに（ファンクションブロックの外から）ユニット番号（&0～15）が、UnitNo（入力変数、データ型 INT）に入力されています。

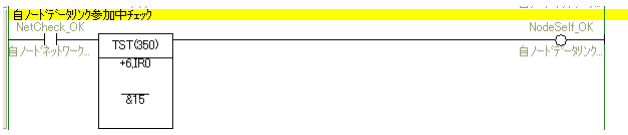
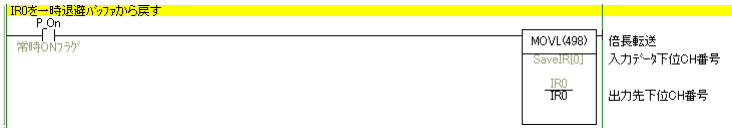
1) UnitNo に &25 を乗算し、Offset（内部変数、データ型 DINT）に格納します。

2) SCPU_relay（内部変数、データ型 WORD、（必要に応じて配列指定、400 要素*1）、AT 指定=1500）の I/O メモリ実効アドレスを、インデックスレジスタ IR0 に格納します。

*1：次の 2) の注のように、SCPU_relay を配列指定して、例えば SCPU_relay [2] と指定することで、1500+（UnitNo×&25）+2CH を指定することも可能です。

3) インデックスレジスタ IR0 内の I/O メモリ実効アドレスに変数 Offset の値（変数 UnitNo×&25）を加算します。

2-5 ファンクションブロック使用のガイドライン
2-5-6 インデックスレジスタの使用方法

2) インデックスレジスタの定数オフセット指定 (割付リレーエリア n+0~n+24CH のいずれかの特定ビットを指定) 	上記処理により、インデックスレジスタ IR0 内に、1500+ (UnitNo×&25) CH の I/O メモリ実効アドレスが格納されているため、 ・CH アドレス：IR0 の定数オフセット指定を使用して、指定します。 例) +2,IR0 と指定すれば、1500+ (UnitNo×&25) +2CH を指定することになります。 - 注：SCPU_relay を配列指定している場合は、SCPU_relay [2] と指定することで、1500+ (UnitNo×&25) +2CH を指定することも可能です。 ・ビットアドレス：CH 内ビットアドレスを指定できる命令（例：TST 命令 /SETB 命令の第 2 オペランド）で指定 例) NetCheck_OK（内部変数、データ型 BOOL）が ON で、かつ IR0 に+6 オフセットした CH 番号 1500+ (UnitNo×&25+6CH) のビット 15 が ON であれば、変数 NodeSelf_OK を ON にします。
●インデックスレジスタを、インデックスレジスタ使用前の内容に戻す 	本ファンクションブロック終了時（またはインデックスレジスタ使用后）にインデックスレジスタの値を元に戻します。 例) 退避させておいた変数 SaveIR[0]の値をインデックスレジスタ IR0 に格納し、本ファンクション開始時（またはインデックスレジスタ使用前）の内容に戻します

2-6 オペランドが複数ワードの先頭/ 最終アドレスを指定している命 令の注意事項

2

仕様

ファンクションブロックをラダー言語でユーザ作成する場合、オペランドが複数ワード（CH）の先頭アドレス（または最終アドレス）を指定している命令のとき、そのオペランドへの変数指定時に、以下に示す注意が必要です。

複数ワード（CH）の先頭アドレス（または最終アドレス）を指定しているオペランドの場合、命令は、AT 指定（または外部変数指定）または内部割当後の「アドレス」にしたがって動作します。したがって、変数のデータ型および配列の要素数は、命令動作とは無関係となります。

必ず、AT 指定の変数か、または命令が処理するデータサイズと一致したサイズの変数を、指定することが必要となります。

⚠ 注意



命令のオペランドが、複数ワード（CH）の先頭アドレス（または最終アドレス）を指定している場合、必ず、AT 指定の変数（または外部変数指定）か、または命令が処理するデータサイズと同一のサイズの変数を、指定してください。以下の点にご注意ください。

1) データサイズが一致していない非配列型変数を AT 指定せずに指定した場合

CX-Programmer がコンパイル時にエラーを出力します。

2) 配列型の変数を指定した場合

●命令が処理をするサイズが固定のオペランドの場合：

配列の要素数は、必ず、命令が処理をするサイズと同一の要素数を、確保してください。さもないと、**CX-Programmer** がコンパイル時にエラーを出力します。

●命令が処理をするサイズが可変のオペランドの場合：

必ず、配列の要素数 $\geq$ 他のオペランドで指定したサイズを確保してください。

・サイズを指定する他のオペランドが、定数の場合：

CX-Programmer Ver.5.0 以降は、コンパイル時にエラーを出力します。

・サイズを指定する他のオペランドが、変数の場合：

配列の要素数が、他のオペランド（変数）で指定したサイズと不一致でも、**CX-Programmer Ver.5.0** 以降は、コンパイル時にエラーを出力しません（ワーニングメッセージは表示します）。

とくに、配列の要素数 $<$ 他のオペランドで指定したサイズの場合（例：命令が処理をするサイズ=16 で、実際に変数テーブル上で登録した要素数=10 の場合）、命令は、その要素数を超えたエリアに対しても、読み書きなどの処理を実行します（例：実際に変数テーブル上で登録した要素数の、次のアドレスの 6CH 分に対しても、読み書きなどの処理をします）。他の命令で、そのエリアを使用していた場合（変数の内部割り当てを含む）、思わぬ動作をし、重大な事故につながる恐れがあります。

複数ワード（CH）の先頭アドレス（または最終アドレス）を指定しているオペランドに、命令が処理するデータサイズとそのサイズが一致していない変数は、使用しないでください。

必ず、命令が必要としているデータサイズと同一のサイズのデータ型の非配列型変数、または命令が必要としているデータサイズと同一のサイズの要素数の配列型の変数を、使用してください。

そうでない場合、以下の結果となります。

- データサイズが一致していない非配列型変数を AT 指定せずに使用した場合
複数ワード (CH) の先頭アドレス (または最終アドレス) を指定しているオペランドに、命令が必要としているデータサイズとそのサイズが一致していないデータ型の非配列型変数を、AT 指定せずに使用した場合、CX-Programmer は、コンパイル時にエラーを出力します。

例) ブロック転送命令: XFER W S D

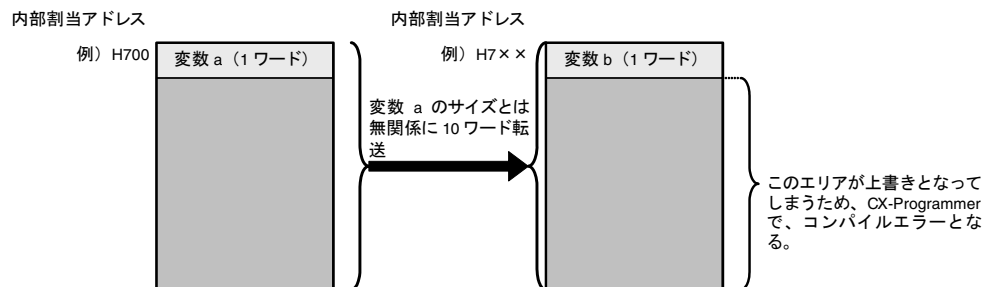
(W: 転送 CH 数、S: 転送元下位 CH 番号、D: 転送先下位 CH 番号)

W に &10、S にデータ型 WORD の変数 a を指定、D にデータ型 WORD の変数 b を指定したとき:

XFER &10 a b

XFER 命令は、変数 a の自動割当されたアドレスを先頭にした 10 ワード分のデータを、変数 b の自動割当されたアドレスを先頭にして、10 ワード分を転送します。そのため、CX-Programmer は、コンパイル時にエラーを出力します。

例) XFER &10 a b
(変数 a、b が、データ型 WORD の場合)



● 配列型の変数を指定した場合

以下の場合に分かれます。

・ 命令が処理するサイズが固定の場合：

命令が処理するサイズが固定のオペランドの場合、そのサイズと配列要素数が一致していないとき、CX-Programmer は、コンパイル時にエラーを出力します。

例) ビット行→ビット列変換命令：COLM S D N の場合

(S：変換データ CH 番号、D：変換結果出力下位 CH 番号、N：ビット位置指定データ)

D には配列要素数=16 のデータ型 WORD の配列を指定するべきところ、もし、配列要素数=10 のデータ型 WORD の配列を指定したとき：

COLM a b[0] c

CX-Programmer は、コンパイル時にエラーを出力します。

・ 命令が処理するサイズが可変の場合：

命令が処理をするサイズが可変のオペランドの場合（その命令の他のオペランドでサイズを指定する場合）、必ず、配列の要素数≧他のオペランドで指定したサイズ（=命令が処理するサイズ）としてください。

・ サイズを指定する他のオペランドが、定数の場合：

CX-Programmer は、コンパイル時にエラーを出力します。

例) ブロック転送命令：XFER W S D の場合

(W：転送 CH 数、S：転送元下位 CH 番号、D：転送先下位 CH 番号)

W に&20、S に要素数=10 のデータ型 WORD の配列型変数 a [要素数 10] を指定、D に要素数=10 のデータ型 WORD の配列型変数 b [要素数 10] を指定した場合 XFER &20 a[0] b[0]

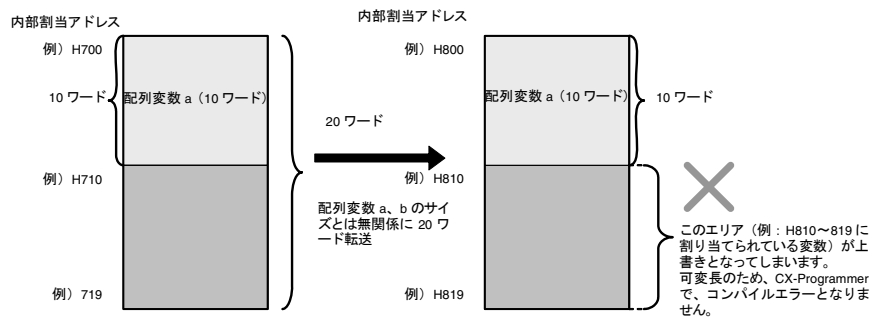
配列型変数 a[0]、b[0]が 10 ワード分の配列であっても、XFER 命令は、W で指定した 20 ワードを転送処理します。そのとき、XFER 命令は、配列型変数が割り当てられた I/O メモリエリアの次のエリアに対して読み書きなどの処理をします。

したがって、もし、a [要素数 10] がアドレス内部割り当て（例：H700～709）、b [要素数 10] もアドレス内部割り当て（例：H800～809）の場合、XFER 命令は、H700～719 のデータを、H800～819 に転送します。

このとき、もし、他の内部割当変数（例:c）が、H810～819 に割り当てられている場合、その変数に対して上書き処理をしてしまい、思わぬ動作をする恐れがあります。

もし 20 ワードを転送する場合は、必ず、配列型変数 a、b の要素数をともに 20 にしてください。

例) XFER &20 a[0] b[0]
(変数 a、b が、要素数 10 の WORD 型の配列の場合) → 20 ワード転送時は、必ず、配列型変数 a、b の要素数を 20 にしてください。



・サイズを指定する他のオペランドが、変数の場合：

万一、配列の要素数が、他のオペランド（変数）で指定したサイズ（=命令が処理するサイズ）と不一致でも、CX-Programmer は、コンパイル時にエラーを出しません。

命令は、配列型変数の要素数とは無関係に、他のオペランドで指定したサイズにしたがって、動作してしまいますので、ご注意ください。

とくに、配列の要素数<他のオペランド（変数）で指定したサイズ（=命令が処理するサイズ）の場合、他の変数に影響を与え、思わぬ動作をする恐れがあります。

2-7 命令の使用可否とオペランドごとの使用変数の制限一覧

■ 命令の使用可否

- ・ CS/CJ/NSJ シリーズ CPU ユニット、CP シリーズ CPU ユニット、FQM1 シリーズユニットの命令の使用可否は、CX-Programmer の命令語ヘルプまたは各 PLC のコマンドリファレンスマニュアルを参照してください。

■ オペランドの使用変数の制限

- ・ 複数ワード (CH) の先頭アドレス (または最終アドレス) を指定するオペランドを持つ命令を使用する場合は、「2-6 オペランドが複数ワードの先頭/最終アドレスを指定している命令の注意事項」の内容をよく確認してから使用してください。
- ・ 命令のオペランドに、複数ワード (CH) の先頭アドレス (または最終アドレス) を指定している場合、その変数に対して、入力パラメータによる受け渡しをすることはできません。その場合、AT 指定、入出力変数または内部変数の配列指定を使用してください。
 - ・ 入出力変数の配列指定を使用するときは、入力パラメータにその先頭アドレスを設定してください。
 - ・ 内部変数の配列指定を使用するときは、必要サイズの要素数の配列指定変数を用意し、ファンクションブロック定義内で各配列にデータをセットした上で、配列型変数の先頭 (または最終) をオペランドに指定してください。
- ・ ネットワーク上の相手の I/O メモリアドレスを指定することが必要なオペランドには、AT 指定の変数を使用してください。

2-8 CPU ユニットのファンクションブロックの仕様

CS/CJ/CP シリーズのファンクションブロックの仕様を示します。その他の事項は、CS/CJ/CP シリーズの各マニュアルを参照してください。

2
仕様

2-8-1 仕様

● CJ2H CPU ユニット

項目	仕様				
形式	CJ2H-CPU68(-EIP)	CJ2H-CPU67(-EIP)	CJ2H-CPU66(-EIP)	CJ2H-CPU65(-EIP)	CJ2H-CPU64(-EIP)
入出力点数	2560				
プログラム容量 (ステップ)	400K	250K	150K	100K	50K
データメモリ	32K ワード (DM/EM ビット参照が可能)				
拡張データメモリ	32K ワード×25 バンク E0_00000～E18_32767	32K ワード×15 バンク E0_00000～EE_32767	32K ワード×10 バンク E0_00000～E9_32767	32K ワード×4 バンク E0_00000～E3_32767	32K ワード×4 バンク E0_00000～E3_32767
強制 S/R 可能 エリア	EM11～EM18	EM7～EME	EM6～EM9	EM3	EM3
ファンク ションブ ロック	定義最大数	1024			
	インスタンス 最大数	2048			
ソース/ コメント エリア	変数テーブル ／コメント文 ／プログラム インデックス	3.5M バイト (*1)			

*1. 格納されるデータによって、メモリ容量の制約はありません。
ソース／コメントエリア合計で 3.5M バイトです。

● CJ2M CPU ユニット

項目	仕様				
形式	CJ2M-CPU11/31	CJ2M-CPU12/32	CJ2M-CPU13/33	CJ2M-CPU14/34	CJ2M-CPU15/35
入出力点数	2560				
プログラム容量 (ステップ)	5K	10K	20K	30K	60K
データメモリ	32K ワード				
拡張データメモリ	32K ワード×1 バンク E0_00000～E0_32767			32K ワード×4 バンク E0_00000～E3_32767	
ファン クシ ョンブ ロッ ク	定義最大数	256			2048
	インスタンス 最大数	256			2048
ソースコ メントメ モリエリ ア	変数テーブル ／コメント文 ／プログラム インデックス	1M バイト (*1)			

*1. 格納されるデータによって、メモリ容量の制約はありません。ソース／コメントエリア
合計で 1M バイトです。

・ ファンクションブロックの消費エリア

CJ2 CPU ユニットでは、ファンクションブロック使用時の消費エリアは、機種によっ
て、下表のように異なります。

CJ2M CPU ユニットは CJ2H CPU ユニットと異なり、ファンクションブロック専用の
エリアとして、「FB プログラムエリア」を用意しています。

機種	形式	ファンクションブロック使用時の消費エリア
CJ2H	形 CJ2H-CPU6□-EIP 形 CJ2H-CPU6□	ユーザプログラムのプログラム容量を消費し ます。
CJ2M	形 CJ2M-CPU3□ 形 CJ2M-CPU1□	ファンクションブロック専用の FB プログラ ムエリアを消費します。 それをオーバーした時には、ユーザプログラ ムのプログラム容量を消費します。

2-8 CPUユニットのファンクションブロックの仕様

2-8-1 仕様

● CS1-H CPU ユニット

項目		仕様								
形式		CS1H-CPU67H	CS1H-CPU66H	CS1H-CPU65H	CS1H-CPU64H	CS1H-CPU63H	CS1G-CPU45H	CS1G-CPU44H	CS1G-CPU43H	CS1G-CPU42H
入出力点数		5120						1280	960	
プログラム容量 (ステップ)		250K	120K	60K	30K	20K	60K	30K	20K	10K
データメモリ		32K ワード								
拡張データメモリ		32K ワード×13 バンク E0_00000～ EC_32767	32K ワード×7 バンク E0_00000～ E6_32767	32K ワード×3 バンク E0_00000～ E2_32767	32K ワード×1 バンク E0_00000～E0_32767		32K ワード×3 バンク E0_00000～ E2_32767	32K ワード×1 バンク E0_00000～E0_32767		
ファンクションブロック	定義最大数	1024	1024	1024	1024	128	1024	1024	128	128
	インスタンス最大数	2048	2048	2048	2048	256	2048	2048	256	256
コメントメモリ ユニット Ver.4.0以降	各ファイル合計 (K バイト)	2048	2048	1280	1280	1280	1280	704	704	704
コメントメモリ内 Ver.3.0以降	FB プログラムメモリ (K バイト)	1664	1664	1024	512	512	1024	512	512	512
	コメントファイル (K バイト)	128	128	64	64	64	64	64	64	64
	プログラムインデックスファイル (K バイト)	128	128	64	64	64	64	64	64	64
	変数テーブル (K バイト)	128	128	128	64	64	128	64	64	64

● CS1D-H CPU ユニット

項目		仕様			
形式		CS1D-CPU68HA	CS1D-CPU67HA	CS1D-CPU67SA	CS1D-CPU44SA
入出力点数		5120			1280
プログラム容量 (ステップ)		400K	250K	250K	30K
データメモリ		32K ワード			
拡張データメモリ		32K ワード×25 バンク E0_00000～E18_32767	32K ワード×13 バンク E0_00000～EC_32767	32K ワード×13 バンク E0_00000～EC_32767	32K ワード×1 バンク E0_00000～E0_32767
ファンクションブロック	定義最大数	1024			
	インスタンス最大数	2048			
コメントメモリ	各ファイル合計 (K バイト)	2048			704
コメントメモリ内	FB プログラムメモリ (K バイト)	1664			512
	コメントファイル (K バイト)	128			64
	プログラムインデックスファイル (K バイト)	128			64
	変数テーブル (K バイト)	128			64

● CJ1-H CPU ユニット

項目		仕様							
形式		CJ1H-CPU67H-R/CPU67H	CJ1H-CPU66H-R/CPU66H	CJ1H-CPU65H-R/CPU65H	CJ1H-CPU64H-R	CJ1G-CPU45H	CJ1G-CPU44H	CJ1G-CPU43H	CJ1G-CPU42H
入出力点数		2560				1280	960		
プログラム容量 (ステップ)		250K	120K	60K	30K	60K	30K	20K	10K
データメモリ		32K ワード							
拡張データメモリ		32K ワード×13 バンク E0_00000～ EC_32767	32K ワード×7 バンク E0_00000～ E6_32767	32K ワード×3 バンク E0_00000～ E2_32767	32K ワード×1 バンク E0_00000～ E2_32767	32K ワード×3 バンク E0_00000～ E2_32767	32K ワード×1 バンク E0_00000～E0_32767		
ファンクションブロック	定義最大数	1024	1024	1024	1024	1024	1024	128	128
	インスタンス最大数	2048	2048	2048	2048	2048	2048	256	256
コメントメモリ ユニット Ver.4.0以降	各ファイル合計 (K バイト)	2048	2048	1280	1280	1280	704	704	704
コメントメモリ内 Ver.3.0以降	FB プログラムメモリ (K バイト)	1664	1664	1024	512	1024	512	512	512
	コメントファイル (K バイト)	128	128	64	64	64	64	64	64
	プログラムインデックスファイル (K バイト)	128	128	64	64	64	64	64	64
	変数テーブル (K バイト)	128	128	128	64	128	64	64	64

● CJ1M CPU ユニット

項目		仕様					
		内蔵入出力機能搭載機種			内蔵入出力機能なし機種		
形式		CJ1M-CPU23	CJ1M-CPU22	CJ1M-CPU21	CJ1M-CPU13	CJ1M-CPU12	CJ1M-CPU11
入出力点数		640	320	160	640	320	160
プログラム容量 (ステップ数)		20K	10K	5K	20K	10K	5K
増設ラック数		最大 1	増設不可		最大 1	増設不可	
データメモリ		32K ワード					
拡張データメモリ		なし					
パルス起動時間		46 μs (加減速なし) 70 μs (加減速あり)		63 μs (加減速なし) 100 μs (加減速あり)	-		
定時割込点数		2		1	2		1
PWM 出力点数		2		1	なし		
サブルーチン番号の 最大値		1024		256	1024		256
JMP 命令のジャンプ番号 の最大値		1024		256	1024		256
内蔵入力		10 点 ・ 割込入力 (パルスキャッチ) 4 点 ・ 高速カウンタ入力 2 点 (位相差 50kHz または単相 100kHz)			-		
内蔵出力		6 点 ・ パルス出力 2 点 100kHz ・ PWM 出力 2 点		6 点 ・ パルス出力 2 点 100kHz ・ PWM 出力 1 点	-		
ファンクション ブロック	定義最大数	128					
	インスタンス 最大数	256					
コメントメモリ ユニット Ver.4.0以降	各ファイル合 計 (K バイト)	704					
コメントメモリ内 ユニット Ver.3.0	FB プログラム メモリ (K バ イト)	256					
	コメントファ イル (K バイ ト)	64					
	プログラムイ ンデックスフ ァイル (K バ イト)	64					
	変数テーブル (K バイト)	64					

● CP1H CPU ユニット

項目		仕様		
		X タイプ CP1H-X□□□□□-□	XA タイプ CP1H-XA□□□□□-□	Y タイプ CP1H-Y□□□□□-□
入出力点数		320 (=内蔵 40 点 + 拡張 40 点 × 7 台)		300 (=内蔵 20 点 + 拡張 40 点 × 7 台)
プログラム容量 (ステップ)		20K		
データメモリ		32K ワード D0~D32767		
拡張 I/O 接続可能数		7 台 (CP シリーズ 拡張 (I/O) ユニット)		
ファン クシ ョ ン ブ ロ ッ ク	定義最大数	128		
	インスタンス 最大数	256		
コメン トメモ リ内	FB プログラ ムメモリ (K バイト)	256		
	コメントファ イル (K バイ ト)	64		
	プログラムイ ンデックスフ ァイル (K バ イト)	64		
	変数テーブル (K バイト)	64		

2-8 CPU ユニットのファンクションブロックの仕様

2-8-1 仕様

● CP1L CPU ユニット

項目		仕様					
形式	Mタイプ	Lタイプ					
	形 CP1L-M60D□-□	形 CP1L-M40D□-□	形 CP1L-M30D□-□	形 CP1L-L20D□-□	形 CP1L-L14D□-□	形 CP1L-L10D□-□	
入出力点数	180 点 (内蔵 60 点+拡張 40 点×3 台)	160 点 (内蔵 40 点+拡張 40 点×3 台)	150 点 (内蔵 30 点+拡張 40 点×3 台)	60 点 (内蔵 20 点+拡張 40 点×3 台)	54 点 (内蔵 14 点+拡張 40 点×1 台)	10 点	
プログラム容量 (ステップ)	10K			5K			
データメモリ	32K ワード D0～D32767			10K ワード D0～D9999、D32000～D32767			
拡張 I/O 接続可能数	3 台 (CP シリーズ 拡張 (I/O) ユニット)			1 台 (CP シリーズ 拡張 (I/O) ユニッ ト)		0 台	
ファンク ション ブロッ ク	定義最大数	128					
	インスタンス 最大数	256					
コメン トメモ リ内	FB プログラ ムメモリ (K バイト)	256					
	コメントファ イル (K バイ ット)	64					
	プログラムイン デックスフ ァイル (K バ イト)	64					
	変数テーブル (K バイト)	64					

項目		仕様	
形式		EM タイプ	EL タイプ
		形 CP1L-EM40D□-□	形 CP1L-EL20D□-□
入出力点数		160 点 (内蔵 40 点+拡張 40 点×3 台)	60 点 (内蔵 20 点+拡張 40 点×3 台)
プログラム容量 (ステップ)		10K	5K
データメモリ		32K ワード D0～D32767	10K ワード D0～D9999、D32000～D32767
拡張 I/O 接続可能数		3 台 (CP シリーズ 拡張 (I/O) ユニット)	1 台 (CP シリーズ 拡張 (I/O) ユニット)
ファンクションブロック	プログラム容量 (ステップ)	10K	
	定義最大数	128	
	インスタンス最大数	256	
コメントメモリ内	FB プログラムメモリ (K バイト)	256	
	コメントファイル (K バイト)	64	
	プログラムインデックスファイル (K バイト)	64	
	変数テーブル (K バイト)	64	

● CP2E CPU ユニット

項目		仕様		
形式	E タイプ	CP2E-E□□DR-A		N タイプ
		CP2E-S□□D□-□		CP2E-N□□D□-□
プログラム容量 (ステップ)	4K		8K	10K
データメモリ	4K ワード D0～D4095		8K ワード D0～D8191	16K ワード D0～D16383
ファン クシ ョン ブ ロ ック	プログラム容量 (ステップ)	4K		10K
	定義最大数	64		
	インスタンス最大数	128		
コメン トメモ リ内	FB プログラムメモリ (K バイト)	256		
	コメントファイル (K バイト)	64		
	プログラムインデックスファイル (K バイト)	64		
	変数テーブル (K バイト)	64		

● NSJ シリーズ

形式	形 NSJ5-TQ0□-G5D、形 NSJ5-SQ0□-G5D、形 NSJ8-TV0□-G5D、形 NSJ10-TV0□-G5D、形 NSJ12-TS0□-G5D		形 NSJ5-TQ0□-M3D、形 NSJ5-SQ0□-M3D、形 NSJ8-TV0□-M3D
入出力点数	1280		640
プログラム容量 (ステップ)	60K		20K
データメモリ	32K ワード		
拡張データメモリ	32K ワード×3 バンク E0_00000～E2_32767		なし
ファンクションブロック	定義最大数	1024	128
	インスタンス最大数	2048	256
コメントメモリ内	FB プログラムメモリ (K バイト)	1024	256
	コメントファイル (K バイト)	64	64
	プログラムインデックスファイル (K バイト)	64	64
	変数テーブル (K バイト)	128	64

● FQM1 シリーズ ユニット Ver.3.0 以降

タイプ	コーディネータモジュール		モーションコントロールモジュール	
形式	形 FQM1-CM002		形 FQM1-MMA22	形 FQM1-MMP22
最大入出力点数	344 点 (=内蔵 24 点 + 基本 I/O ユニット 320 点)		内蔵 20 点	
プログラム容量 (ステップ)	10K			
データメモリ	32K ワード			
ファンクションブロック	定義最大数	128		
	インスタンス最大数	256		
コメントメモリ内	FB プログラムメモリ (K バイト)	256		
	コメントファイル (K バイト)	64		
	プログラムインデックスファイル (K バイト)	64		
	変数テーブル (K バイト)	64		

2-8-2 タイマ系命令の動作について

PLCのプロパティに、「T2048-4095のTIM実行を他のTIMと同じにする」のチェックがあります。(CP2Eにはありません)

このチェック設定に応じて、タイマ系命令の動作が以下ようになります。

● チェックをONに設定した場合

タイマ番号T0~4095の動作はすべて同じで、以下ようになります。

・タイマ番号T0~4095のタイマ動作

更新タイミング	内容
各命令の実行時	・ 命令実行ごとに現在値を更新します。 ・ 現在値=0でタイムアップフラグをON。0でなければ、タイムアップフラグをOFFにします。
全タスクの実行終了時	1サイクルに1回ごとに、現在値を更新します。
80msごとの更新	サイクルタイムが80msを超える場合、80msごとに現在値を更新します。

● チェックをOFFに設定した場合（デフォルトではOFF）

タイマ番号T0~2047とT2048~4095で、以下のように現在値更新タイミングが異なります。

この動作は、CS/CJシリーズ CS1-H/CJ1-H/CJ1M CPU ユニット ユニット Ver.2.0 以前と同一です（CS/CJシリーズコマンドリファレンスの各タイマ系命令の説明内の参考を参照）。

・タイマ番号T0~2047のタイマ動作

更新タイミング	内容
各命令の実行時	・ 命令実行ごとに現在値を更新します。 ・ 現在値=0でタイムアップフラグをON。0でなければ、タイムアップフラグをOFFにします。
全タスクの実行終了時	1サイクルに1回ごとに、現在値を更新します。
80msごとの更新	サイクルタイムが80msを超える場合、80msごとに現在値を更新します。

・タイマ番号T2048~4095のタイマ動作

更新タイミング	内容
各命令の実行時	・ 命令実行ごとに現在値を更新します。 ・ 現在値=0でタイムアップフラグをON。0でなければ、タイムアップフラグをOFFにします。
全タスクの実行終了時	現在値を更新しません。
80msごとの更新	サイクルタイムが80msを超えても、現在値を更新しません。

したがって、ファンクションブロックの変数はデフォルトではタイマ番号T3072~4095を使用しているため、タイマの内部割当範囲をデフォルトで使用する場合は、PLCのプロパティの「T2048-4095のTIM実行を他のTIMと同じにする」のチェックをONに設定してください。

2-9 ファンクションブロック使用時のプログラムステップ数とインスタンス実行時間について

2-9-1 ファンクションブロック使用時の、プログラムステップ数について

■ ファンクションブロックの消費容量（ステップ数）

ファンクションブロックを使用すると、以下の2つで、プログラム容量（ステップ数）を消費します。

- ①ファンクションブロック定義の容量
- ②それをユーザプログラム上にインスタンスとして貼り付けたときのインスタンス生成処理容量

したがって、②の貼り付ける数（インスタンス数）が多ければ多いほど、容量（ステップ数）を多く消費する点にご注意ください。

■ ファンクションブロック使用時のプログラムステップ数の目安

ファンクションブロック定義を作成し、ユーザプログラム上にインスタンスを貼り付けたときの、プログラムステップ数は、以下を目安としてください。

$$\begin{aligned} & \text{ステップ数} \\ &= \text{インスタンス数} \times (\text{呼び出し部容量 } m + \text{入出力パラメータ転送部容量 } n \times \text{パラメータ数}) \\ & + \text{ファンクションブロック定義内命令ステップ数 } p \text{ (注)} \end{aligned}$$

注：同一ファンクションブロック定義を複数貼り付けた場合（インスタンスが複数の場合）、2個目以降のインスタンスでは、ファンクションブロック定義内命令ステップ数（p）は、消費されません。そのため、上記計算式では、ファンクションブロック定義内命令ステップ数（p）に対してインスタンス数を乗じていません。

内 容			使用ステップ数
m	呼び出し部		57 ステップ
n	入出力パラメータ転送部 () 内は、データ型	1 ビット（BOOL）の入力変数、出力変数の場合	6 ステップ
		1 ワード（INT、UINT、WORD）の入力変数、出力変数の場合	6 ステップ
		2 ワード（DINT、UDINT、DWORD、REAL）の入力変数、出力変数の場合	6 ステップ
		4 ワード（LINT、ULINT、LWORD、LREAL）の入力変数、出力変数の場合	18 ステップ
		入出力変数の場合	6 ステップ
p	ファンクションブロック定義内命令ステップ数	命令ステップ数の合計（通常のユーザプログラムと同じ）に、27 ステップを加算したステップ数	

2-9 ファンクションブロック使用時のプログラムステップ数とインスタンス実行時間について

2-9-1 ファンクションブロック使用時の、プログラムステップ数について

例)

データ型が1ワード (INT) の入力変数=5 個、

データ型が1ワード (INT) の出力変数=5 個、

ファンクションブロック定義部分のステップ数が100 ステップの場合：

1つのインスタンスのステップ数= $57 + (5+5) \times 6$ ステップ+100 ステップ+27 ステップ
= 244 ステップです。

また、ST 言語で記述された場合は実ステップ数の計算はできません。

ファンクションブロック定義内の命令ステップ数は、ファンクションブロック定義のプロパティで確認することができます。

2-9-2 ファンクションブロック使用時のインスタンス実行時間について

ファンクションブロック定義を作成し、ユーザプログラム上にインスタンスを貼り付けた場合、インスタンス実行によるサイクルタイムの影響時間は、以下のとおりです。

$$\begin{aligned} & \text{インスタンスによるサイクルタイムの影響時間} \\ &= \text{起動時間 A} \\ & \quad + \text{入出力パラメータ転送処理時間 B} \\ & \quad + \text{ファンクションブロック定義内命令実行時間 C} \end{aligned}$$

各 A、B、C の時間は、以下のとおりです。

内容			CPU ユニット形式						
			形 CJ1H-CPU6□H-R 形 CJ2H-CPU6□(-EIP)	形 CJ2M-CPU□□	形 CS1H-CPU6□H 形 CJ1H-CPU6□H	形 CS1G-CPU4□H 形 CJ1G-CPU4□H NSJ	形 CJ1M-CPU□□	形 CS1D-CPU6□HA	形 CS1D-CPU□□SA
A	起動時間	入出力パラメータ転送を除く起動時間	3.3 μs	7.4 μs	6.8 μs	8.8 μs	15.0 μs	8.0 μs	6.9 μs
B	入出力パラメータ転送処理時間 () 内は、データ型	1 ビット (BOOL) の入力変数、出力変数の場合	0.24 μs	0.88 μs	0.4 μs	0.7 μs	1.0 μs	0.6 μs	0.5 μs
		1 ワード (INT、UINT、WORD) の入力変数、出力変数の場合	0.19 μs	0.88 μs	0.3 μs	0.6 μs	0.8 μs	0.5 μs	0.4 μs
		2 ワード (DINT、UDINT、DWORD、REAL) の入力変数、出力変数の場合	0.19 μs	1.2 μs	0.5 μs	0.8 μs	1.1 μs	0.7 μs	0.7 μs
		4 ワード (LINT、ULINT、LWORD、LREAL) の入力変数、出力変数の場合	0.38 μs	2.96 μs	1.0 μs	1.6 μs	2.2 μs	2.4 μs	2.2 μs
		入出力変数の場合	0.114 μs	0.4 μs	0.4 μs	0.5 μs	1.2 μs	0.7 μs	0.6 μs
C	ファンクションブロック定義内命令実行時間	命令処理時間の合計 (通常のユーザプログラムと同じ)							

2-9 ファンクションブロック使用時のプログラムステップ数とインスタンス実行時間について
2-9-2 ファンクションブロック使用時のインスタンス実行時間について

内容			CPU ユニット形式		
			形 CP1H-X □□□□-□ 形 CP1H-XA □□□□-□ 形 CP1H-Y □□□□-□	形 CP1L-M □□□□-□ 形 CP1L-L □□□□-□	形 CP2E-N □□D□-□ 形 CP2E-S □□□□-□ 形 CP2E-E □□□□-□
A	起動時間	入出力パラメータ転送を除く起動時間	15.0 μs	320.4 μs	54.37 μs
B	入出力パラメータ転送処理時間 () 内は、データ型	1 ビット (BOOL) の入力変数、出力変数の場合	1.0 μs	59.52 μs	9.74 μs
		1 ワード (INT、UINT、WORD) の入力変数、出力変数の場合	0.8 μs	13.16 μs	5.4 μs
		2 ワード (DINT、UDINT、DWORD、REAL) の入力変数、出力変数の場合	1.1 μs	15.08 μs	6.31 μs
		4 ワード (LINT、ULINT、LWORD、LREAL) の入力変数、出力変数の場合	2.2 μs	30.16 μs	20.9 μs
		入出力変数の場合	— (未サポート)	— (未サポート)	— (未サポート)
C	ファンクションブロック定義内命令実行時間	命令処理時間の合計 (通常のユーザプログラムと同じ)			

例) 形 CJ1H-CPU67H-R で、
データ型が 1 ワード (INT) の入力変数=3 個、
データ型が 1 ワード (INT) の出力変数=2 個、
ファンクションブロック定義部分の命令処理時間の合計が 10 μs の場合：
1 つのインスタンス実行時間= 3.3 μs + (3+2) × 0.19 μs + 10 μs = 14.25 μs です。

注：同一ファンクションブロック定義を複数貼り付けた場合 (インスタンスが複数の場合)、インスタンス数に応じて、実行時間が増加します。

ファンクションブロック編

第3章 ファンクション
ブロックの作成

3-1 使用手順フロー

以下のような手順でファンクションブロックを作成し、ファイルに保存、実機 PLC へ転送、モニタ／デバッグをします。

ファンクションブロックを作成

1. プロジェクトを作成する

■ 新規プロジェクトを作成する場合

- 1) CX-Programmer を起動し、[ファイル] メニューから [新規作成] を選択
- 2) PLC 機種として、機種名 CS1H-H、CS1G-H、CJ1G-H、CS1D-H、CS1D-S、CJ1H-H、CJ1M、CP1H、CP1L、NSJ、FQM1-CM (MMA/MMP) を選択

■ CX-Programmer の既存プロジェクトを再利用する場合

- 1) CX-Programmer を起動し、[ファイル] メニューから既存の CX-Programmer プロジェクトファイル (.cxp) *1 を読み出します。
- 2) PLC 機種として、機種名 CS1H-H、CS1G-H、CS1D-H、CS1D-S、CJ1G-H、CJ1H-H、CJ1M、CP1H、CP1L、NSJ、FQM1-CM (MMA/MMP) を選択

…3-2-1 項参照

2. ファンクションブロック定義を新規作成 (挿入) する

- 1) [ファンクションブロック] を選択して右クリック
- 2) ポップアップメニューから [ファンクションブロック挿入] | [ラダー] または [ストラクチャードテキスト] を選択

…3-2-2 項参照

3. ファンクションブロック定義を作成する (ユーザ作成の場合のみ)

■ 変数の登録の後、ラダープログラムまたは ST プログラムを入力する場合

- 1) 「変数テーブル」に変数を登録
- 2) ラダープログラムまたは ST プログラム作成

■ ラダープログラムまたは ST プログラムを入力する都度、変数を登録する場合

- 1) ラダープログラムまたは ST プログラム作成
- 2) 都度「変数テーブル」に変数を登録

…3-2-3 項参照

4. ファンクションブロック定義をプログラム上に貼り付ける (インスタンスを生成する)

■ ラダーセクションウィンドウ上に挿入して、インスタンス名を入力する方法

- 1) ファンクションブロックのインスタンス (コピー) を作成したい位置にカーソルを置いて、[F] キーを押す
- 2) インスタンス名を入力
- 3) コピーしたいファンクションブロック定義を選択

■ あらかじめグローバル変数テーブル上にインスタンス名を登録した後、そのインスタンス名を選択する方法

- 1) グローバル変数テーブル上に、変数のデータ型として「FUNCTION BLOCK」を選択
- 2) ラダーセクションウィンドウ上で [F] キーを押す
- 3) [FB インスタンス] フィールドのプルダウンリストから登録済みのインスタンス名を選択

…3-2-4 項参照

5. ファンクションブロックを外部と入出力割付する ([Enter] キーでパラメータを設定する)

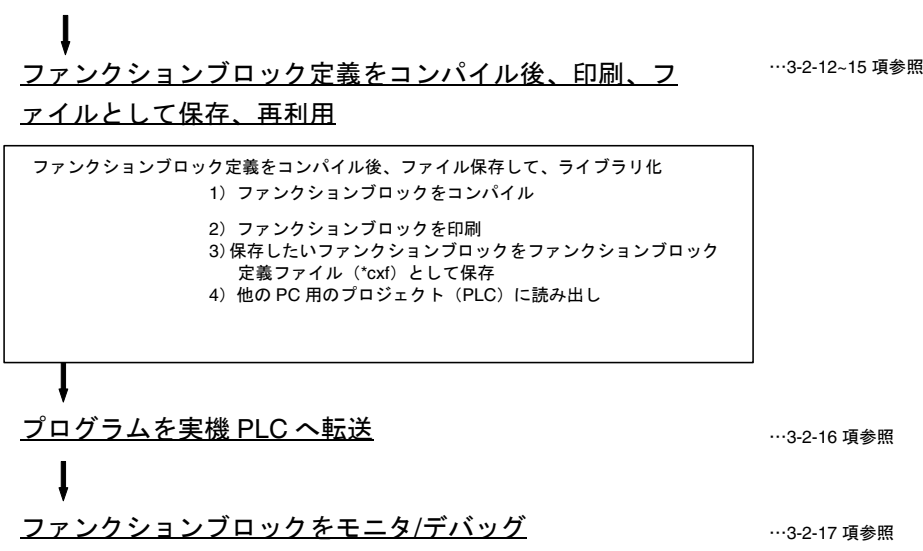
- 1) 入力変数または出力変数の位置にカーソルを置いて、[Enter] キーを押す
- 2) 入力変数への受け渡し元アドレスを入力、また出力変数からの受け渡し先アドレスを入力

…3-2-5 項参照

6. 変数の内部割当範囲 (FB インスタンスエリア) を設定

- 1) インスタンスを選択し、[PLC] メニューから [ファンクションブロック/SFC メモリ] - [ファンクションブロック/SFC アドレス割付] を選択
- 2) FB インスタンスエリアを設定

…3-2-6 項参照



3-2 操作方法

3-2-1 プロジェクトを作成する

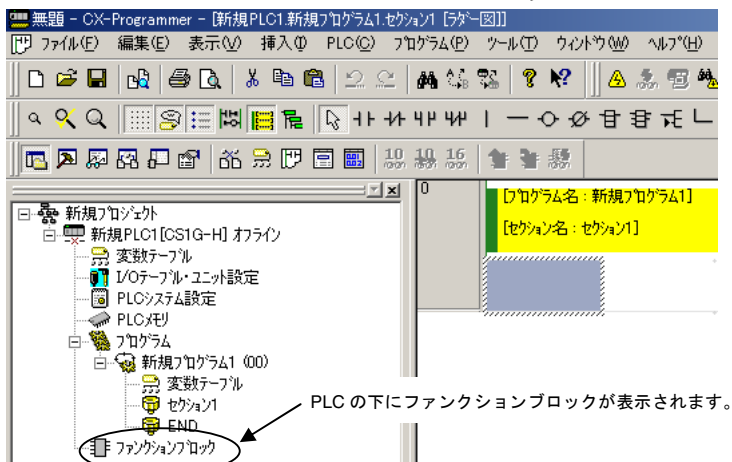
- 1 CX-Programmer を起動し、[ファイル] メニューから [新規作成] を選択します。
- 2 [PLC 機種変更] ダイアログボックスにて、[PLC 機種] リストから、ファンクションブロックが使用可能な以下の PLC 機種のいずれかを選択します。

PLC 機種	CPU タイプ
CJ2H	CPU68/67/66/65/64/68-EIP/67-EIP/66-EIP/65-EIP/64-EIP
CJ2M	CPU11/12/13/14/15/31/32/33/34/35
CS1G-H	CPU45H/44H/43H/42H
CS1H-H	CPU67H/66H/65H/64H/63H
CS1D-H	CPU68A/67A
CS1D-S	CPU67A/44A
CJ1G-H	CPU45H/44H/43H/42H
CJ1H-H	CPU67H-R/66H-R/65H-R/64H-R/67H/66H/65H
CJ1M	CPU23/22/21/13/12/11
CP1H	XA/X/Y
CP1L	M/L
CP1L-E	EM/EL
CP2E	E/S
CP2E-N	N
NSJ	G5D (形 NSJ5-TQ0□-G5D、形 NSJ5-SQ0□-G5D、形 NSJ8-TV0□-G5D、形 NSJ10-TV0□-G5D、形 NSJ12-TS0□-G5D で、共通) M3D (形 NSJ5-TQ0□-M3D、形 NSJ5-SQ0□-M3D、形 NSJ8-TV0□-M3D で、共通)
FQM1-CM	CM002
FQM1-MMA	MMA22
FQM1-MMP	MMP22

- 3 [設定] ボタンを押して CPU タイプを選択します。
それ以外の設定は、CX-Programmer のオペレーションマニュアル (SBCA-337) を参照してください。

3-2-2 ファンクションブロック定義を新規作成 (挿入) する

- 1 プロジェクトが新規作成されると、ワークスペース上に、以下のように、[ファンクションブロック] アイテムが、アイコン  の後に表示されます。



- 2 以降、この [ファンクションブロック] アイテムの下に、ファンクションブロック定義を、挿入する形で、新規作成します。

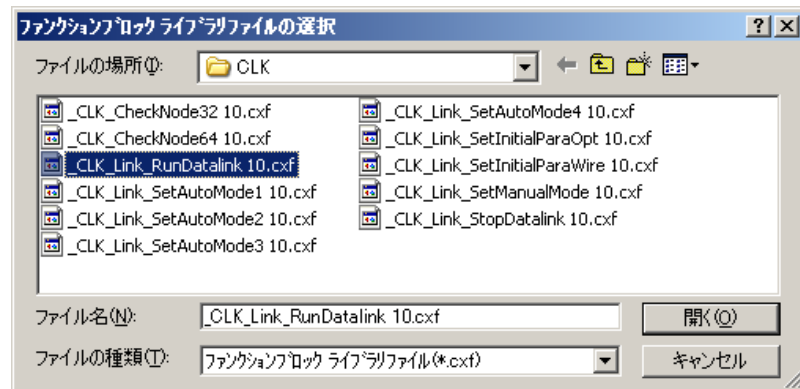
3-2 操作方法

3-2-2 ファンクションブロック定義を新規作成（挿入）する

● Smart FB ライブラリファイルを使用する場合

Smart FB ライブラリファイル（.cxf）を挿入します。

1. ワークスペース上の［ファンクションブロック］を選択して、右クリックして、ポップアップメニューから［ファンクションブロック挿入］の下の［ライブラリファイルからロード］を選択します。（または、［挿入］メニューから［ファンクションブロック］ | ［ライブラリファイルからロード］を選択します。）
2. 以下の［ファンクションブロックライブラリファイルの選択］ダイアログボックスが表示されます。



参 考

［ファンクションブロックライブラリファイルの選択］ダイアログボックスでのデフォルトのフォルダ（「ファイルの場所」）を、［ツール］ | ［オプション］の［全般］タブの［FB ライブラリの格納場所］にて、指定できます。

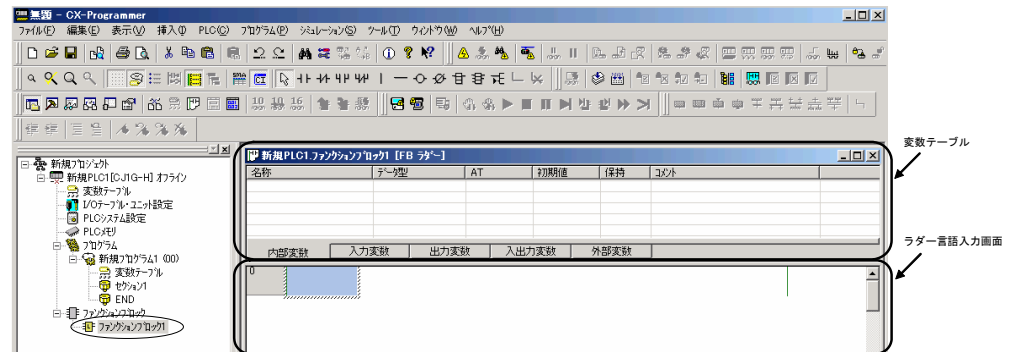
3. Smart FB ライブラリファイルがあるフォルダを指定し、ライブラリファイルを選択して、［開く］ボタンをクリックします。
アイコンの後にライブラリファイルが、ファンクションブロック定義として挿入されます。

■ ファンクションブロック定義の内容

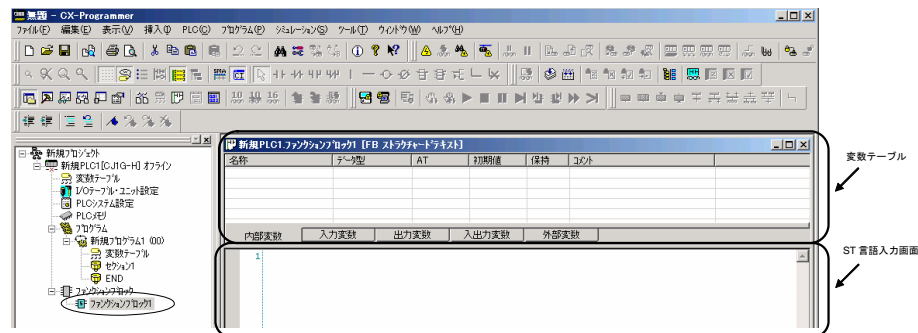
● ファンクションブロック定義をユーザが作成する場合

新規作成した「ファンクションブロック 1」をダブルクリックする（または右クリックしてポップアップメニューから「開く」を選択する）と、以下のように、右側ビューの上側にファンクションブロックで使用する変数テーブル（「変数」のテーブル）が表示され、右側ビューの下側に、ラダー言語入力画面または ST 言語入力画面が表示されます。

・ ラダー言語の場合



・ ST 言語の場合



このように、ファンクションブロック定義は、インタフェースとしての「変数」のテーブルと、アルゴリズムとしてのラダー言語または ST 言語から成ります。

・ インタフェースとしての「変数」のテーブル：

この時点ではまだ、変数に PLC の I/O メモリアドレスは割り付いていません。

・ アルゴリズムとしてのラダー言語または ST 言語：

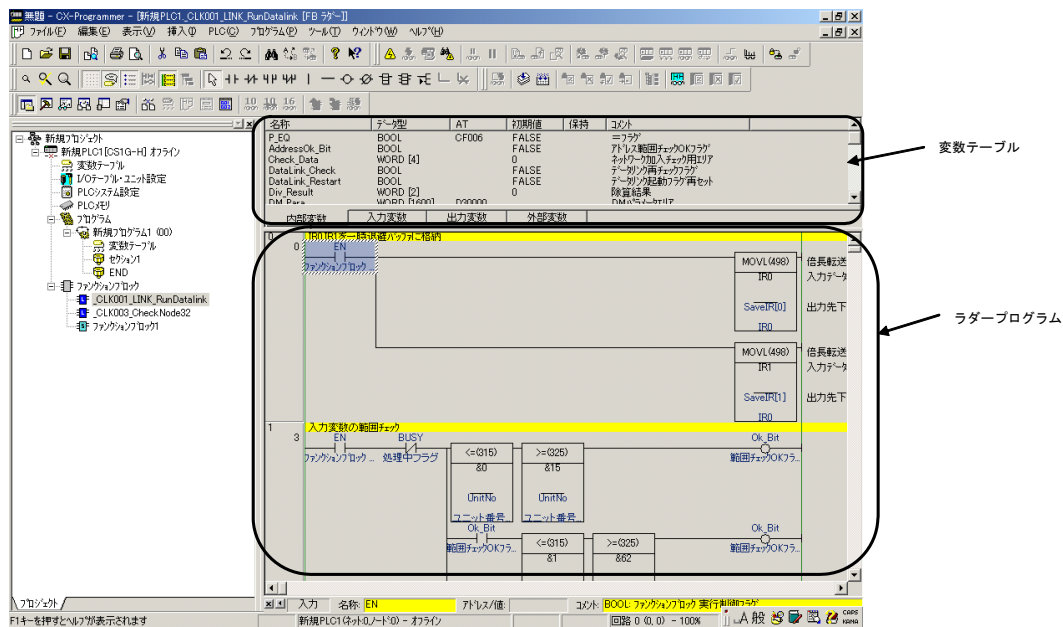
- ・ ファンクションブロックのラダー言語には、通常のプログラムで使用できる命令を使用することができます。ただし、一部使用できない命令があり、それらの命令については 2-3 項「制限事項」を参照してください。
- ・ ファンクションブロックの ST 言語には、IEC 61131-3 で定義されている ST 言語を入力することができます。

3-2 操作方法

3-2-2 ファンクションブロック定義を新規作成（挿入）する

● Smart FB ライブラリファイルを使用する場合

挿入した Smart FB ライブラリをダブルクリックする（または右クリックしてポップアップメニューから「開く」を選択する）と、以下のように、右側ビューの上側に作成済みの変数テーブルが表示され、右側ビューの下側に、作成済みのラダープログラムが表示されます。いずれもグレーで表示され、編集することはできません。



注：Smart FB ライブラリファイル（.cxf）の場合、ファンクションブロック定義は、デフォルトでは、表示不可となっています。表示したい場合は、ファンクションブロックのプロパティにて、「内部のプログラムを表示可能にする」のチェックボックスをチェックしてください（操作：プロジェクトワークスペース上の Smart FB ライブラリファイルを右クリックして、ポップアップメニューから「プロパティ」を選択して、「一般」タブにて設定してください）。

3-2-3 ファンクションブロック定義を作成する（ユーザ作成の場合のみ）

ファンクションブロック定義の内容の作成方法には、以下の2通りがあります。

- ・変数の登録の後、ラダー言語またはST言語を入力する方法
- ・ラダー言語またはST言語を入力する都度、変数を登録する方法

■ 変数の登録の後、ラダー言語またはST言語を入力する場合

1) 「変数テーブル」に変数を登録

変数テーブルは、変数の種類ごとに、以下の4つのシートに分かれています。

内部変数、入力変数、出力変数、入出力変数、外部変数

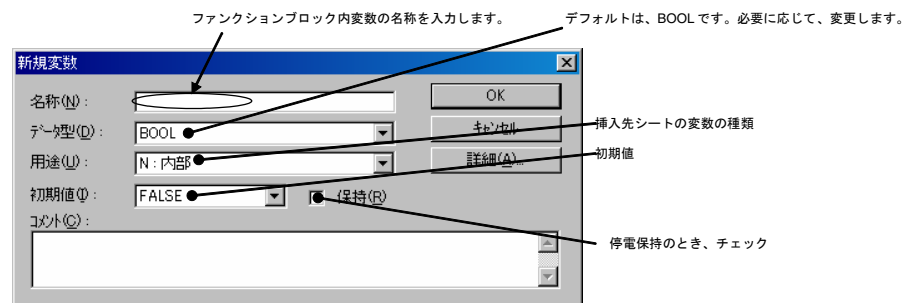
したがって、変数を登録/表示するとき、変数の種類ごとに登録/表示するシートを切り替えます。

1. 上側の「変数テーブル」の内、挿入したい編集の種類のシートをアクティブ*1にします。そこにカーソルを置いて、右クリックしてポップアップメニューから、
 - －変数を最終行に追加するときは、[変数追加]を選択します。
 - －変数を途中の行の上側または下側に追加するときは、[変数挿入] | [上側] または [下側] を選択します。

*1: 変数入力時に、用途（N：内部、I：入力、O：出力、E：外部、P：入出力）を切り替えることで、登録するシートを切り替えることもできます。

以下の「新規変数」ダイアログボックスが表示されます。

- ・名称：変数名を入力します。
- ・データ型：データタイプを選択します。
- ・用途：変数の種類を選択します。
- ・初期値：変数の初期値（運転開始時の値）を選択します。
- ・保持：ここをチェックすると、電源OFF→ON時、または運転開始（「プログラム」→「モニタ」または「運転」モードへ変更）時にも、前回の値を保持します。チェックをしない場合、初期化されます。

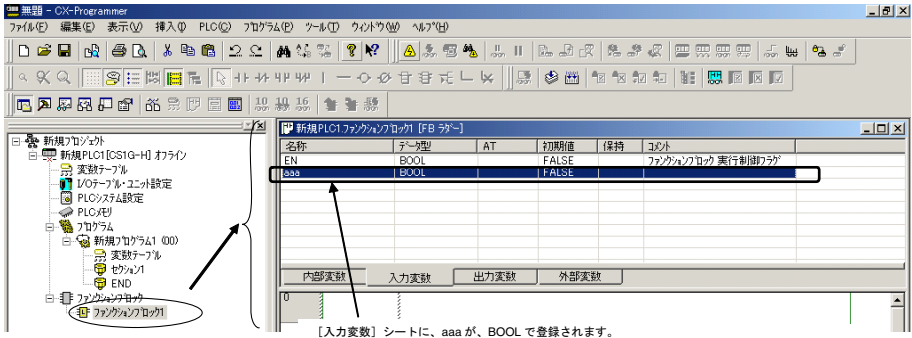


- 注：・ユーザ定義の外部変数の場合、同一変数名をグローバル変数テーブルに登録することで、グローバル変数テーブルを参照することができます。
- ・システム定義の外部変数は、あらかじめ外部変数のテーブルに登録されています。

3-2 操作方法

3-2-3 ファンクションブロック定義を作成する（ユーザ作成の場合のみ）

- 例えば、変数名に“aaa”と入力し、[OK] ボタンをクリックします。
以下のように、「変数テーブル」の「入力変数」シートに変数名“aaa”が BOOL
で登録されます。

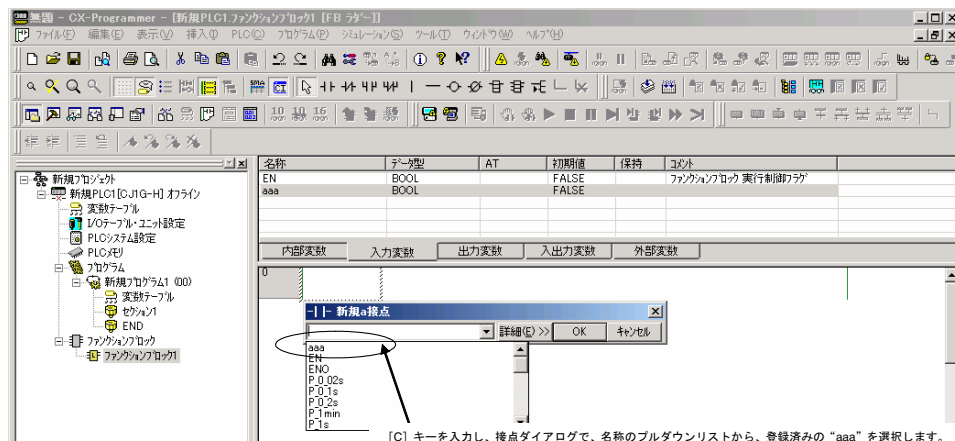


- 注1：変数を追加した後、変数を指定して反転表示させた上で、ドラッグ&ドロップすることで、変数の行位置を移動させることもできます。
ただし、「名称」以外の列にカーソルを置いて、ドラッグ&ドロップすることが必要です。
- 注2：変数入力後に、ダブルクリックして、「用途」フィールド（N：内部、I：入力、O：出力、E：外部、P：入出力）を切り替えることで、登録するシートを移動させることもできます。
また、「内部変数」、「入力変数」、「出力変数」、「入出力変数」、「外部変数」の各シート間で、変数を、コピーまたは移動させることができます。変数を選択し、右クリックしてポップアップメニューから「コピー」または「切り取り」をし、「貼り付け」を選択することによります。
- 注3：AT（実アドレスへの割り当て）指定をする変数の場合でも、変数名を入力することが必要です。
- 注4：ファンクションブロックの変数テーブルには、「変数名」として、I/O メモリのアドレスを示す、以下の文字を入力することはできません。
A、W、H（またはHR）、D（またはDM）、E（またはEM）、T（またはTIM）、C（またはCNT）を先頭としてその後に数値（チャンネル番号）

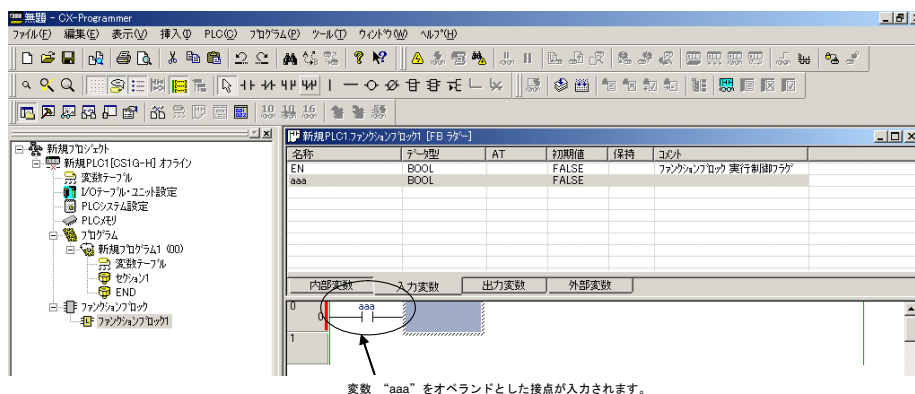
2) アルゴリズム作成

● ラダー言語の場合（クラシックモード）

1. [C] キーを入力し、接点ダイアログで、名称のプルダウンリストから、登録済みの“aaa”を選択します。



2. [OK] ボタンをクリックすると、次のように、変数“aaa”（変数の種類：入力変数）をオペランドとした接点が入力されます。



以降、CX-Programmerでの通常のラダープログラム作成とまったく同様な操作です。
 注：ファンクションブロック内では、命令のオペランドに、アドレスを直接入力することはできません。

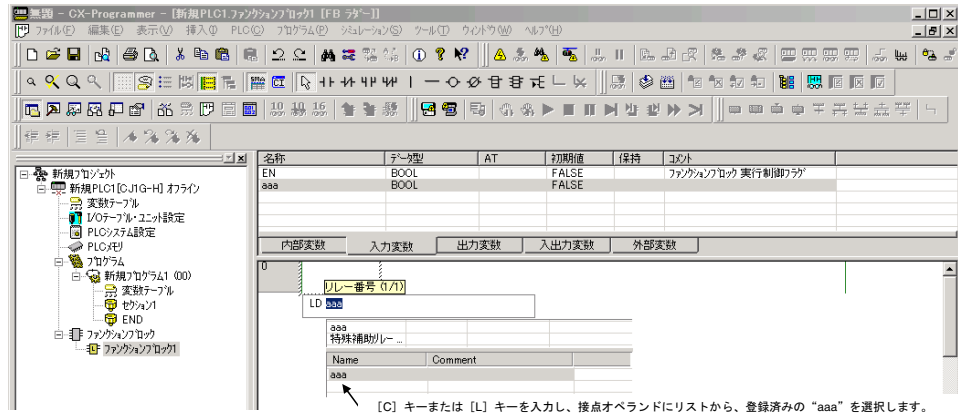
インデックスレジスタ（IR）/データレジスタ（DR）のみは、直接（変数としてでなく）アドレス「DR0~5」、「IR0~15」（直接指定時）、「,IR0~15」（間接指定時）を入力できます。

● ラダー言語の場合（スマートインプットモード）

1. [C] または [L] キーを入力し、オペランドにリストから、登録済みの“aaa”を選択します。

3-2 操作方法

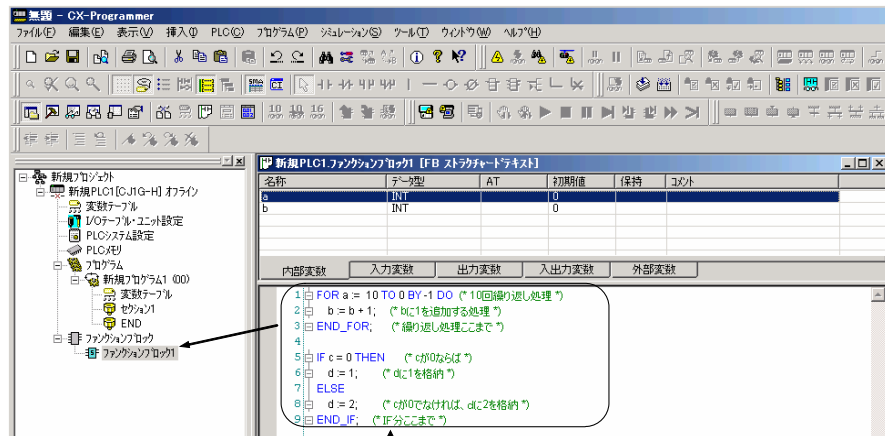
3-2-3 ファンクションブロック定義を作成する（ユーザ作成の場合のみ）



2. [ENTER] キーを押すと、変数“aaa”（変数の種類：入力変数）をオペランドとした接点が入力されます。

● ST 言語の場合

1. ST 言語入力画面に直接 ST 言語*1 を記述するか、または汎用のテキストエディタで作成した ST 言語*1 のテキストを [編集] メニューから [貼り付け] を選択して貼り付けます。
*1：IEC 61131-3 に準拠しています。詳細は、ストラクチャードテキスト編 第5章の「ST 言語の仕様」を参照。



ST 言語を直接入力するか、またはテキストエディタで作成した ST 言語のテキストを貼り付けます。

- 注1：タブまたはスペースを入力して、インデントをつけることが可能です（アルゴリズムに影響がありません）。メニューおよびツールバーからも可能です。
- 注2：ST 入力画面にテキストを入力または貼り付けると、文のキーワード（予約語）は青字、コメントは緑字、エラーは赤字、その他は黒字で、自動的に表示されます。
- 注3：フォントサイズ、表示色を変更したい場合は、[ツール] メニューから [オプション] を選択して、[表示] タブで、[ST フォント] ボタンをクリックします。
フォント名、フォントサイズ（デフォルトは9ポイント）、色を、変更できます。
- 注4：ST 言語の仕様について詳細は、ストラクチャードテキスト編 第5章「ST 言語の仕様」を参照してください。

■ ラダー言語または ST 言語を入力する都度、変数を登録する場合

ラダー言語または ST 言語の入力を先行することもできます。

● ラダー言語の場合

ラダー言語の場合、未登録の変数名を入力する都度、その変数登録を促すダイアログボックスが表示されます。そこで変数を登録します。

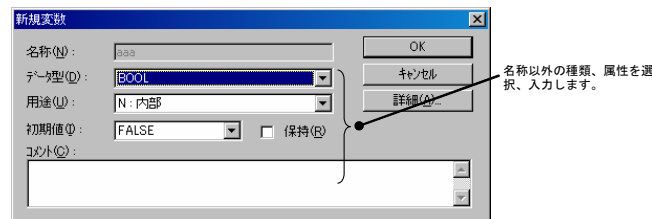
以下の手順によります。

1. [C] キーを入力し、接点ダイアログで、名称に（未登録の）例えば“aaa”を入力します。

注：ファンクションブロック内では、命令のオペランドに、アドレスを直接入力することはできません。

ただし、インデックスレジスタ（IR）/データレジスタ（DR）のみは、直接（変数としてでなく）アドレス「DR0~5」、「IR0~15」（直接指定時）、「,IR0~15」（間接指定時）を入力できます。

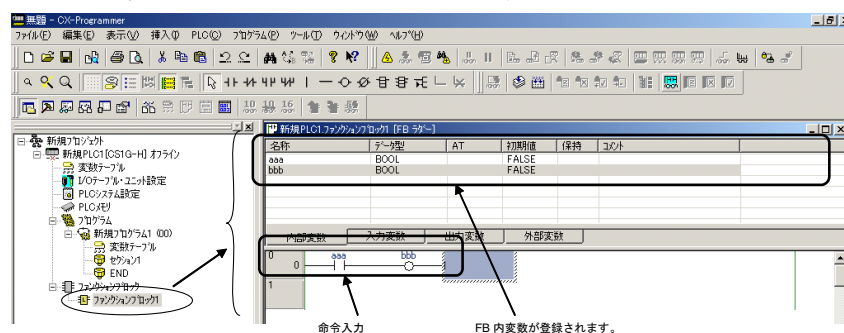
2. [OK] ボタンをクリックすると、[新規変数] ダイアログボックスが表示されます。（応用命令の場合、オペランド数分の [新規変数] ダイアログボックスが表示されます。）



その場合、入力した変数はすべていったん、以下の条件で表示されます。

- ・用途： “内部変数”
- ・データ型： 接点は BOOL、CH は WORD
- ・初期値： デフォルト
- ・保持： なし

3. 必要に応じて、変更をした上で、[OK] ボタンをクリックします。
4. 以下のように、入力した変数が、上側の「変数テーブル」に登録されます。



- 5 入力した変数の種類および属性が正しくない場合は、必要に応じて、「変数テーブル」の変数をダブルクリックして、修正します。

3-2 操作方法

3-2-3 ファンクションブロック定義を作成する（ユーザ作成の場合のみ）

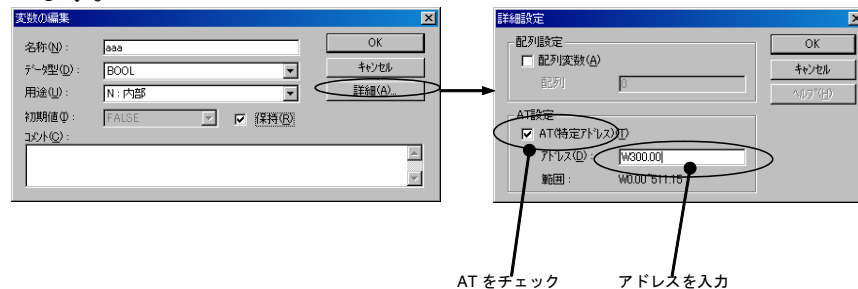
参 考

●AT（実アドレスへの割り当て）指定について

基本 I/O ユニット/高機能ユニットの割付アドレスを指定したい場合や、**CX-Programmer** に登録されていない特殊補助リレーを指定したい場合は、変数属性の **AT**（実アドレスへの割り当て）指定をします。ただし、その場合でも、変数名の入力は必要です。

AT 指定をする場合、以下の操作を実行します。

1. 新規変数] ダイアログボックスにて変数名を入力後、[詳細] ボタンをクリックします。[詳細設定]ダイアログボックスが表示されます。
2. **AT** 設定フィールドの **AT**（特定アドレス）をチェックして、実アドレスを入力します。



なお、ファンクションブロック定義のアルゴリズム内で、**AT** 指定の変数を記述するときも、（**AT** 指定なしの変数と同様に）変数名で指定します。

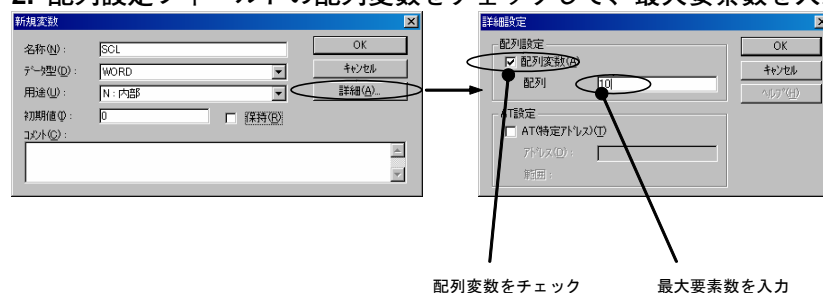
例) 変数名 **Restart**、**AT** 指定=**A501.00** のとき、命令のオペランド内では、**Restart** と指定します。

●配列指定について

同一のデータ属性の複数の変数を使用する場合で、かつそれらを 1 つのグループとして管理したい場合、変数属性の配列指定をします。

配列指定をする場合、以下の操作を実行します。

1. 新規変数] ダイアログボックスにて変数名を入力後、[詳細] ボタンをクリックします。[詳細設定]ダイアログボックスが表示されます。
2. 配列設定フィールドの配列変数をチェックして、最大要素数を入力します。



なお、ファンクションブロック定義のアルゴリズム内で配列変数を記述するときは、変数名の後に、添字を半角 `[]` で囲んで指定します。

例) 変数名 **PV**、最大要素数=3 のとき、命令のオペランド内では、**PV[0]**、**PV[1]**、**PV[2]**と指定します。

添字の指定方法には、以下の 3 つの方法があります。

- ・ 添字を直接番号で指定する方法（ラダーまたは **ST** 言語）：上記例で、**PV[1]**
- ・ 添字を変数で指定する方法（ラダーまたは **ST** 言語）：上記例で、**PV[a]**（**a** はデータ型 **INT** の変数名）
- ・ 添字を式で指定する方法（**ST** 言語のみ可能）：上記例で、**PV[a+b]**、**PV[a+1]** など（**a** はデータ型 **INT** の変数名）

● ST 言語の場合

ST 言語を先に記述したとき、変数テーブルへの入力を促すダイアログボックスは表示されません。したがって、ST 言語を記述した後、かならず、その都度または一括で、使用した変数名を変数テーブルへ登録してください（上側の変数テーブルの入力したいシートにカーソルを置いて、右クリックしてポップアップメニューから、[変数挿入] を選択することによる）。

注：ST 言語の仕様について詳細は、ストラクチャードテキスト編 第 5 章「ST 言語の仕様」を参照してください。

■ ユーザプログラム側の回路をコピーし、ファンクションブロック定義のラダープログラム上に貼り付ける場合

ユーザプログラム側の 1 つまたは複数の回路をコピーして、ファンクションブロック定義のラダープログラム上に貼り付けることもできます。

ただし、以下の点に注意が必要です。

● 貼り付け元の命令のオペランドが、「アドレスのみ」の場合

アドレスは、ファンクションブロック定義の変数テーブルには登録されません。

貼り付け後、オペランド上に赤字で表示されます。命令をダブルクリックして、オペランドに変数名を入力し直してください。

注：例外的に、インデックスレジスタ（IR）、データレジスタ（DR）は、貼り付け後、オペランドとしてそのまま機能します。

● 貼り付け元の命令のオペランドが、「アドレス+I/O コメント」の場合

- ・ [ツール] | [オプション] の [変数] タブの [変数名を自動的に生成する] をチェックしたとき

ユーザプログラム側の変数名（グローバル変数テーブル上のみ）が、自動的に、「AutoGen_」 + 「アドレス」となります（チェックをはずすと、すべて変数名なしに戻ります）。

例 1：アドレスが 100.01 の場合、変数名が AutoGen_100_01 となります。

例 2：アドレスが D0 の場合、変数名が AutoGen_D0 となります。

ユーザプログラム上の回路をコピーして、そのままファンクションブロック定義のプログラムに貼り付ければ、（回路のコピーと同時に、）「AutoGen_アドレス」を「変数名」として、また、I/O コメントを「コメント」として、変数が、ファンクションブロック定義の変数テーブルに自動的に登録されます。

これにより、アドレス+I/O コメントでプログラミングした回路を、ファンクションブロックで再利用することが、容易となります。

注：インデックスレジスタ（IR）、データレジスタ（DR）は、元々グローバル変数テーブルに登録できず、AutoGen_は付きません。

- ・ [ツール] | [オプション] の [変数] タブの [変数名を自動的に生成する] をチェックしないとき

アドレスおよび I/O コメントは、ファンクションブロック定義の変数テーブルには登録されません。アドレスがオペランド上に赤字で表示されます。I/O コメントは失われます。命令をダブルクリックして、オペランドに変数名を入力し直してください。

注：例外的に、インデックスレジスタ（IR）、データレジスタ（DR）は、貼り付け後、オペランドとしてそのまま機能します。

3-2 操作方法

3-2-3 ファンクションブロック定義を作成する（ユーザ作成の場合のみ）

● 貼り付け元の命令のオペランドが、「変数」の場合

ユーザプログラム側の変数は、ファンクションブロック定義の変数テーブルの「内部変数」に、自動的に登録されます。ただし、以下の点に注意が必要です。

・ アドレスについて

変数のアドレスは、登録されません。もし、同一のアドレスを指定する場合は、AT 指定をすることが必要です。

・ データ型の変換について

ユーザプログラムをファンクションブロック定義に貼り付けたとき、以下のように、変数のデータ型が変換されます。

ユーザプログラム上の変数のデータ型	→	ファンクションブロックのプログラムに貼り付けたときの変数のデータ型
CHANNEL		WORD
NUMBER		変数は登録されず、値（番号）がオペランドに、直接、定数として、貼り付けられます。
UNIT BCD		WORD
UDINT BCD		DWORD
ULINT BCD		LWORD

ただし、データ型が CHANNEL、NUMBER、UNIT BCD、UDINT BCD、ULINT BCD の上記変数の場合、（プログラムでなく）変数テーブルから変数をコピーして、ファンクションブロック定義内の変数テーブルに貼り付けようとした場合、貼り付けることができません。

注：変数名を自動生成した変数（変数名：「AutoGen_」 + 「アドレス」）をグローバル変数テーブルからコピーして、ファンクションブロック定義の変数テーブル上に貼り付けることはできません。

■ ユーザプログラム側の回路の一部をファンクションブロック定義として生成する場合

ユーザプログラム側の 1 つまたは複数の回路を、ファンクションブロック定義（ラダー）として生成することもできます。

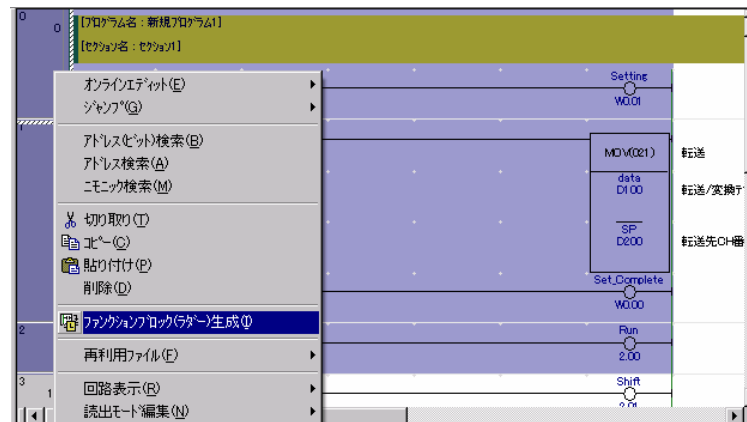
お願い

本機能は、完全な自動生成でなく、既存のラダープログラムをもとにファンクションブロック定義を作成するときに、それを支援する機能です。

本機能によってファンクションブロック定義を生成後、必ず、[FB 変数割付] ダイアログボックスおよびアウトプットウィンドウに表示されるワーニングメッセージをご確認の上、生成されたアルゴリズム（プログラム）をチェックし、必要に応じて修正処理を行ってください。

1. ユーザプログラム側の1つまたは複数の回路を右クリックして、ポップアップメニューから「ファンクションブロック（ラダー）生成」を選択します。

注：データ型ビューに構造体定義が存在する場合は、メニュー「ファンクションブロック（ラダー）生成」を選択することはできません。



2. 以下の「FB 変数割付」ダイアログボックスが表示されます。



選択された回路の命令におけるオペランドのアドレスが、使用状況に応じて、以下のように、内部変数／入力変数／出力変数／入出力変数に、自動的に割り当てられます。

		選択回路内			
		未使用	入力部で使用	出力部で使用	入力部と出力部で使用
選択回路外	未使用 (*1)	—	内部変数	内部変数	内部変数
	使用	—	入力変数	出力変数	入出力変数

*1: I/O 割付されている場合でも、選択回路外で使用されていない場合は、未使用とみなし（選択回路内のいずれの位置で使用されていても）、内部変数に変換します。

注1：変数名なしのアドレスの場合、変数名は、自動で、「AutoGen_」+「アドレス」となります。

AT 指定は、自動では「なし」となります。

3. 必要に応じて、内部変数／入力変数／出力変数／入出力変数への割り当てを変更してください。用途の割り当ては、変数を選択し、ポップアップメニューの「用途変更」から変更します。



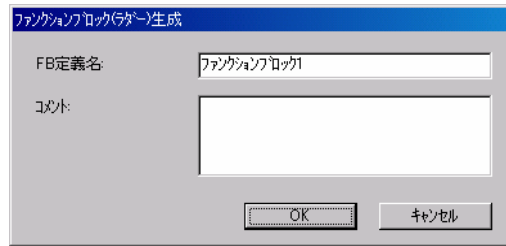
変数リスト上で変数をダブルクリックして、変数名、コメントを変更します。また、配列指定、AT 指定をすることも可能です。

3-2 操作方法

3-2-3 ファンクションブロック定義を作成する（ユーザ作成の場合のみ）

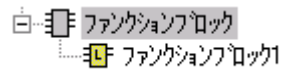
4. [OK] ボタンをクリックします。

以下の「ファンクションブロック（ラダー）生成」ダイアログボックスが表示されます。

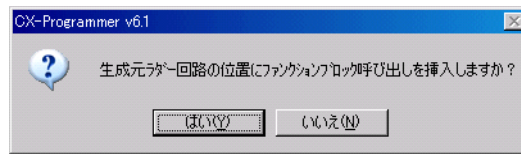


「FB 定義名」、「コメント」を入力し、[OK]ボタンをクリックします。

5. 設定された内容を元に、ファンクションブロック定義が生成され、ワークスペース上の「ファンクションブロック」の下に表示されます。



6. 生成元としたラダー回路の次の位置に、作成したファンクションブロック定義のインスタンス（ファンクションブロック呼び出し）を挿入で貼り付けるか否かを選択する、以下のダイアログボックスが表示されます。



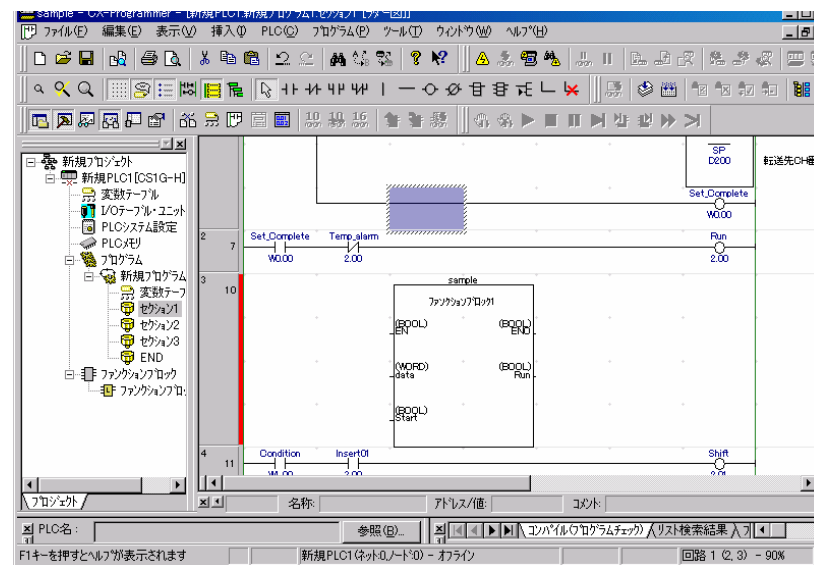
7. 挿入する場合、[はい] ボタンをクリックします。挿入しない場合、[いいえ] ボタンをクリックします。

8. [はい] ボタンをクリックした場合、以下の「新規ファンクションブロック呼び出し」ダイアログボックスが表示されます。



FB インスタンス名を入力して、[OK] ボタンをクリックします。

生成元としたラダー回路の次の回路位置に、インスタンスが以下のように挿入で貼り付けられます。



9. 挿入したインスタンスの入力条件、パラメータを入力します。

参考

本生成機能は、すでに実際のプログラムで使用されている実績のあるラダー回路を元にファンクションブロック化するのに便利な機能です。

内部的には、選択された回路範囲内外での「アドレス」の使われ方を解析し、入力/出力/内部変数/入出力変数の用途を、可能な限り自動判別しています。

したがって、変数のみで「アドレス」がないオペランドを含む回路があると、生成はできません。

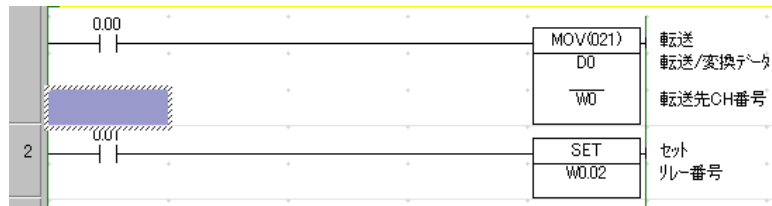
変数のみで「アドレス」がないオペランドを含む回路の場合は、ユーザプログラム側から、ファンクションブロック定義のラダープログラム上へコピー&ペーストを行ってください。（「**ユーザプログラム側の回路をコピーし、ファンクションブロック定義のラダープログラム上に貼り付ける場合**」を参照）

- ファンクションブロック定義作成前に、回路内のプログラム変更が必要なケース
 以下の場合、ファンクションブロック定義作成前に、あらかじめ回路内のプログラム変更が必要となります。
- ・ 選択された回路内で、同一アドレスがともに接点指定とチャネル指定されている場合
 接点とチャネルで、それぞれ別の変数として登録されます。そのため、あらかじめプログラムを変更することにより、別変数となることを回避することが必要です。

3-2 操作方法

3-2-3 ファンクションブロック定義を作成する（ユーザ作成の場合のみ）

例) MOV 命令で W0、SET 命令で W0.02 が指定されている場合：



この場合は、接点指定をチャネル指定となるように、命令を変更する必要があります。

以下のように、事前にともに W0 指定となるように、MOV 命令で W0、SETB 命令で W0 およびビットアドレス&2 を指定します。

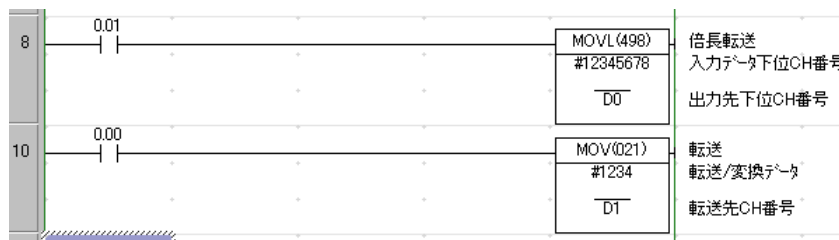


● ファンクションブロック定義をいったん作成した後、オペランド指定変更が必要なケース

以下の場合、ファンクションブロック定義をいったん作成した後、配列変数を使用した、オペランド指定変更が必要となります。

- ・ 選択された回路内で、ある命令で複数チャネルのデータ（例：倍長データ）の先頭 CH を指定しており、他の命令でその複数チャネルのデータの一部（例：ワードデータ）を指定している場合

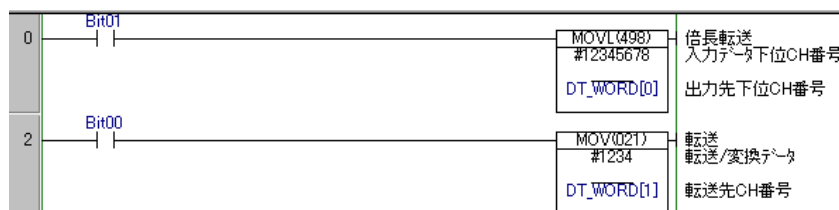
例) MOVL 命令で倍長データの先頭 D0 を指定し、MOV 命令で D1 を指定している場合：



この場合は、ファンクションブロック定義をいったん作成した後で、以下のように、配列指定の先頭要素指定と、任意要素指定に変更してください。

例) DT_WORD は、WORD 型の配列変数、要素数は 2 とします。

その上で、MOVL 命令で DT_WORD[0]を指定し、MOV 命令で DT_WORD[1]を指定します。



- ・ 命令内の2つのオペランドで、先頭チャンネル、最終チャンネルを、それぞれ指定している場合

例) BSET 命令で先頭 D0、最終 D9 を指定している場合：



この場合は、ファンクションブロック定義をいったん作成した後に、以下のように、配列指定での先頭要素指定、最終要素指定に変更してください。

例) DT_WORD は、WORD 型の配列変数、要素数は 10 とします。

その上で、BSET 命令で、先頭 DT_WORD[0]、最終 DT_WORD[9]を指定します。



- ・ 命令のあるオペランドのサイズが他のオペランドの値に応じて変わる場合

例) XFER 命令の転送 CH 数&5、転送元先頭 D0、転送先 D100 を指定している場合：



この場合は、ファンクションブロック定義をいったん作成した後に、以下のように、入力データ下位 CH、出力先下位 CH とともに、配列指定の先頭要素指定に変更してください。

例) D0、DT100 は、ともに WORD 型の配列変数、要素数は 5 とします。

その上で、XFER 命令で、先頭 DT_WORD1 [0]、先頭 DT_WORD2 [0]を指定します。



3-2 操作方法

3-2-4 ファンクションブロック定義をプログラム上に貼り付ける（インスタンスを生成する）

3-2-4 ファンクションブロック定義をプログラム上に貼り付ける（インスタンスを生成する）

グローバル変数テーブルに、インスタンスを登録する場合、以下のいずれかの方法によります。

方法1 ファンクションブロック定義を選択してプログラムに配置するときに、新規にインスタンス名を入力します。このとき、自動的にグローバル変数テーブルに、インスタンスが登録されます。

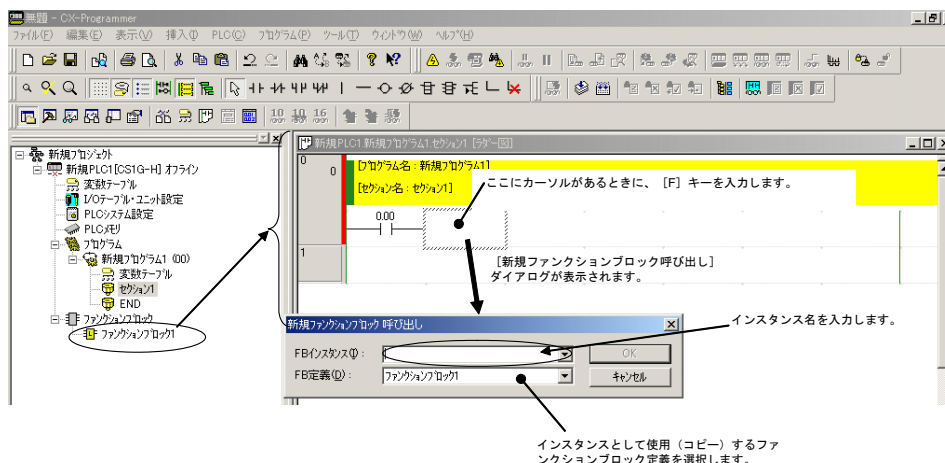
方法2 グローバル変数のデータ型を「FUNCTION BLOCK」として、使用（コピー）するファンクションブロック定義を指定して、インスタンス名を入力して、登録します。

注：ST 言語の場合、変数のデータ型として「FUNCTION BLOCK」を選択し、そのインスタンス名を使用して、ファンクションブロック呼び出し文を記述することで、ファンクションブロックを呼び出すことができます。

ファンクションブロック定義を選択してプログラムに配置するときに、同一のインスタンス名を入力します。

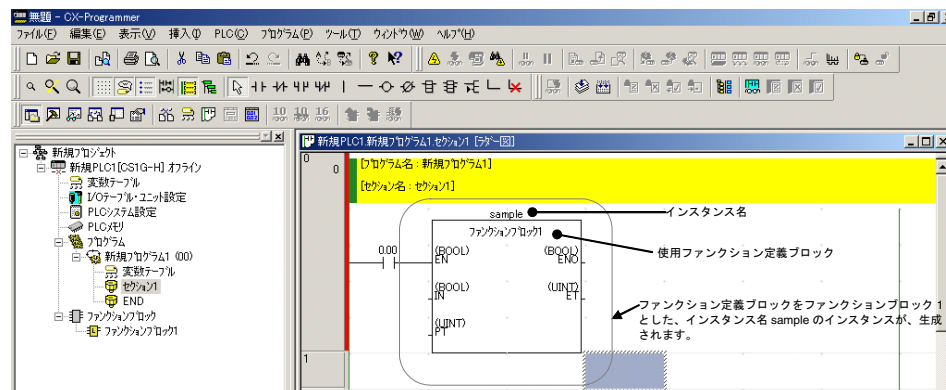
方法1) ラダーセクションウィンドウ上に [F] キーで挿入して、インスタンス名を入力する方法

1. ラダーセクションウィンドウ上で、ファンクションブロックのインスタンス（コピー）を作成したい位置にカーソルを置いて、[F] キーを押します。（または、[挿入] メニューから [ファンクションブロック呼び出し] を選択します。） [ファンクションブロック呼び出し] ダイアログボックスが表示されます。
2. 任意のインスタンス名を入力し、コピーしたいファンクションブロック定義を選択して、[OK] ボタンをクリックします。



3-2-4 ファンクションブロック定義をプログラム上に貼り付ける (インスタンスを生成する)

- 例として、インスタンス名 ([FB インスタンス] フィールド) を **sample** として、ファンクション定義ブロック ([ファンクションブロック定義] フィールド) をファンクションブロック 1 として、 [OK] ボタンをクリックします。
以下のように、ファンクションブロック定義ファンクションブロック 1 をコピーして、インスタンス名 **sample** のインスタンスが生成されます。



- グローバル変数テーブル上に、インスタンス名を名称を **sample**、データ型「FUNCTION BLOCK」としたインスタンスが自動的に登録されます。

方法 2) あらかじめグローバル変数テーブル上にインスタンス名を登録した後、そのインスタンス名を選択する方法

あらかじめグローバル変数テーブル上にインスタンス名を登録した後、そのインスタンス名を選択する形で、インスタンスを生成することもできます。

- ラダー言語の場合、グローバル変数テーブル上に変数のデータ型として「FUNCTION BLOCK」を選択して、ファンクションブロックのインスタンスを挿入して、登録します。
ST 言語の場合、変数のデータ型として「FUNCTION BLOCK」を選択し、そのインスタンス名を使用して、以下のようなファンクションブロック呼び出し文を記述することで、ファンクションブロックを呼び出すことができます。
インスタンス名*1 の後に、 () 内に、引数指定 (自身の入力変数の値を、呼び出し先の入力変数に渡す指定)、同時に、戻り値指定 (呼び出し先の出力変数の値を、自身の出力変数に受ける指定) を記述します。
*1: データ型がファンクションブロックの任意の内部変数名です。
- ラダーセクションウィンドウ上で [F] キーを押します。 [ファンクションブロック呼び出し] ダイアログボックスで [FB インスタンス] フィールドのプルダウンリストから、登録済みのインスタンス名を選択することで、インスタンスを生成することもできます。

3-2 操作方法

3-2-5 ファクションブロックを、外部と入出力割付する（[Enter] キーでパラメータを設定する）

■ 制限事項

インスタンスを生成する場合、以下の制限事項があります。詳細は、2-3 項「制限事項」を参照してください。

- ・1 回路内に複数のインスタンスを配置することはできません。
- ・インスタンスの左側で回路を分岐することはできません。
- ・インスタンスを左母線に直接接続することはできません。つまり、EN には必ず入力が必要です。

参 考

インスタンスを生成後、元となったファンクションブロック定義の、変数テーブル内の入出力変数に対して、変更操作を加えた場合、インスタンスの左母線は、エラーを示す赤表示となります。このとき、そのインスタンスを選択し、右クリックして「ファンクションブロック呼び出しの更新」を選択すると、そのインスタンスに対してファンクションブロック定義の変更内容が更新されます。このとき、エラーを示す左母線の赤表示が解除されます。

3-2-5 ファクションブロックを、外部と入出力割付する（[Enter] キーでパラメータを設定する）

生成されたインスタンスは、入力変数に対して入力パラメータを設定、および出力変数に対して出力パラメータを設定することで、外部との入出力が可能となります。

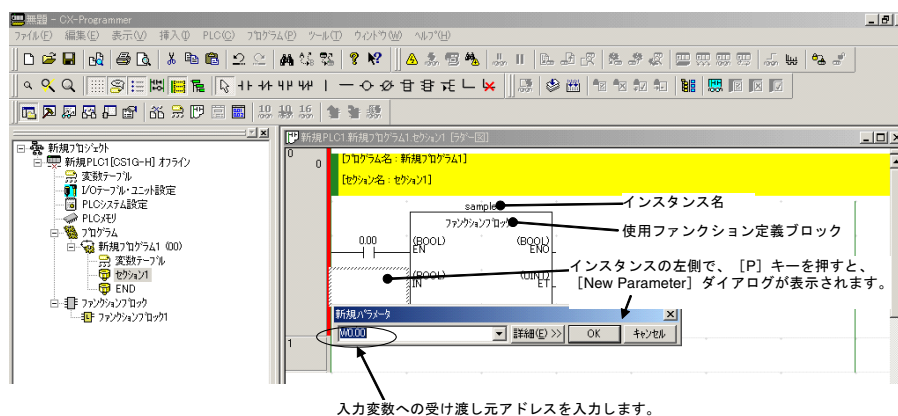
- ・入力パラメータには、値、アドレス、プログラム側の変数（グローバル変数、ローカル変数）*1 を設定可能です。
- ・出力パラメータには、アドレス、プログラム側の変数（グローバル変数、ローカル変数）*2 を設定可能です。

*1：ファンクションブロックの入力変数のデータサイズと、プログラム側の変数のデータサイズが、一致していることが必要です。

*2：ファンクションブロックの出力変数のデータサイズと、プログラム側の変数のデータサイズが、一致していることが必要です。

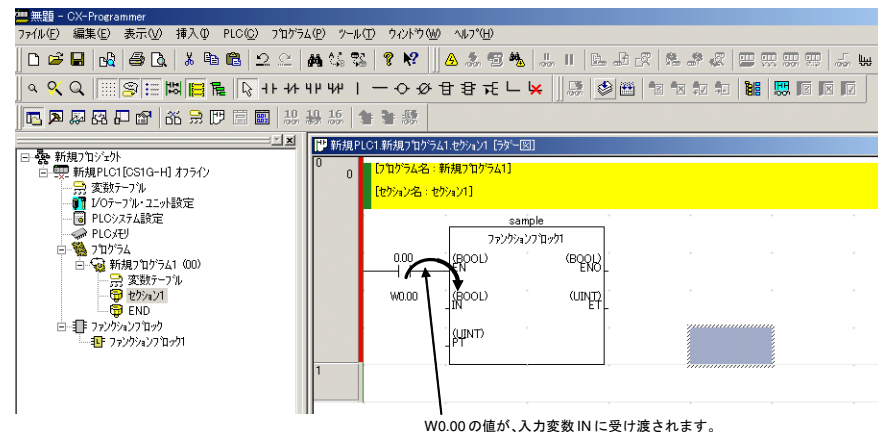
1. 入力変数の左隣の位置、または出力変数の右隣の位置にカーソルを置いて、[Enter] キーを押します（または、[挿入] メニューから「ファンクションブロックパラメータ」を選択します）。

以下のように、「パラメータ編集」ダイアログボックスが表示されます。



3-2-5 ファクションブロックを、外部と入出力割付する（[Enter] キーでパラメータを設定する）

2. 入力変数への受け渡し元アドレスを設定します。また出力変数からの受け渡し先アドレスを設定します。



お願い

すべての入力パラメータに、データを設定してください。ブランクの入力パラメータが1つでもあると、インスタンスの左母線は、エラーを示す赤表示となります。このとき、プログラムを実機 PLC へ転送することができません。

パラメータへの値の入力方法一覧

パラメータに値を入力するときの入力方法は以下のとおりです。

入力変数のデータ型	内容	ビット	パラメータへの値の入力方法	値の最大範囲
BOOL	ブール型	1	P_Off、P_On	0 (FALSE)、1 (TRUE)
INT	整数型	16	+	-32768 ~ +32767
DINT	倍精度整数型	32	-	-2147483648 ~ +2147483647
LINT	4 倍精度整数型	64		-9223372036854775808 ~ +9223372036854775807
UINT	符号なし整数型	16	+	&0 ~ 65535
UDINT	符号なし倍精度整数型	32		&0 ~ 4294967295
ULINT	符号なし 4 倍精度整数型	64		&0 ~ 18446744073709551615
REAL	実数型	32	+	-3.402823×10 ³⁸ ~ -1.175494×10 ⁻³⁸ 、0、+1.175494×10 ⁻³⁸ ~ +3.402823×10 ³⁸
LREAL	倍精度実数型	64	-	-1.79769313486232×10 ³⁰⁸ ~ -2.22507385850720×10 ⁻³⁰⁸ 、0、+2.22507385850720×10 ⁻³⁰⁸ ~ +1.79769313486232×10 ³⁰⁸
WORD	長さ 16 のビット列型	16	#の後に、16 進数（4 桁以下）または &または+の後に 10 進数	#0000~FFFF または&0 ~ 65535
DWORD	長さ 32 のビット列型	32	#の後に、16 進数（8 桁以下）または &または+の後に 10 進数	#00000000~FFFFFFFF または&0 ~ 4294967295
LWORD	長さ 64 のビット列型	64	#の後に、16 進数（16 桁以下）または &または+の後に 10 進数	#0000000000000000~FFFFFFFFFFFFFFFF または&0 ~ 18446744073709551615

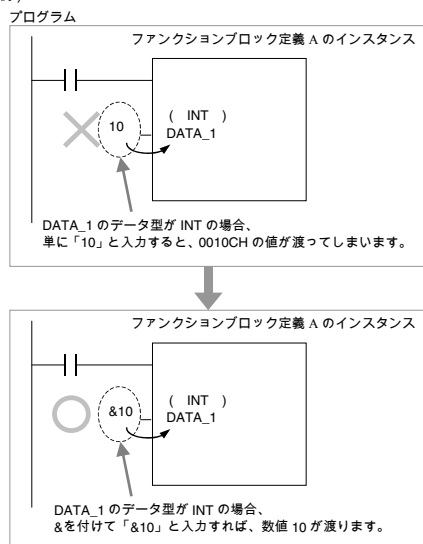
3-2 操作方法

3-2-5 ファンクションブロックを、外部と入出力割付する（[Enter] キーでパラメータを設定する）

お願い

- ・入力変数のデータ型が **BOOL** 型以外の場合、パラメータに、単に数値のみ（例：20）を入力すると、数値ではなく、CIO エリアの CH アドレス（例：0020CH）の値を渡してしまいますので、ご注意ください。
値を設定する場合、単に数値を入力しないで、必ず、&、#、または+、-の接頭辞の後に、数値を入力してください。

例)

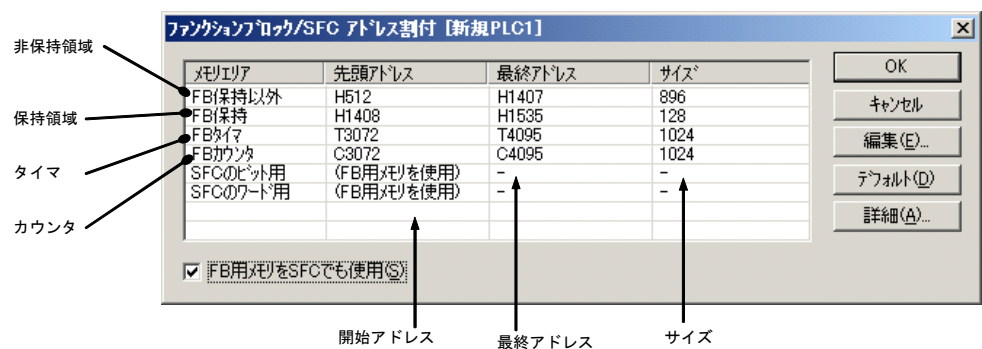


- ・入力変数のデータ型が **BOOL** 型の場合、パラメータに、単に数値のみ（0 または 1）を入力すると、**0000CH** のビット 00（0.00）または **0000CH** のビット 01（0.01）の値を渡してしまいますので、ご注意ください。必ず、0（OFF）は、**P_Off** を、1（ON）は **P_On** を入力してください。

3-2-6 変数の内部割当範囲 (FB インスタンスエリア) を設定する

ファンクションブロックで使用している各変数の使用アドレスを、内部的に割り当てるエリア (FB インスタンスエリア) を設定します。

1. ラダーセクションウィンドウ上のインスタンスを選択 (またはグローバル変数テーブル上のインスタンスを選択) し、[PLC] メニューから [メモリ割付] | [ファンクションブロック/SFC メモリ] | [ファンクションブロック/SFC アドレス割付] を選択します。
2. 以下の [ファンクションブロック/SFC アドレス割付] ダイアログボックスが表示されます。
3. 各変数の使用アドレスを内部的に割り当てるエリア (FB インスタンスエリア) を設定します。



非保持領域、保持領域は、CH (ワード) 単位で設定します。

タイマ領域、カウンタ領域は、タイマ番号、カウンタ番号単位で設定します。

デフォルトでの内部割当範囲：

初期値は以下のとおりとなります。

・CJ2 シリーズ CPU ユニットの場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	サイズ	
保持以外	H512	H1407	896	CIO、WR、HR、DM、EM ^{*1}
保持	H1408	H1535	128	HR、DM、EM ^{*1}
タイマ	T3072	T4095	1024	TIM
カウンタ	C3072	C4095	1024	CNT

*1. 以下の EM バンクを指定した場合、強制セット/リセットが可能となります。

形 CJ2H-CPU64(-EIP)/-CPU65(-EIP)	EM バンク 3
形 CJ2H-CPU66(-EIP)	EM バンク 6～9
形 CJ2H-CPU67(-EIP)	EM バンク 7～E
形 CJ2H-CPU68(-EIP)	EM バンク 11～18

3-2 操作方法

3-2-6 変数の内部割当範囲 (FB インスタンスエリア) を設定する

・ CS/CJ シリーズ CPU ユニット Ver.3.0 以降、NSJ の場合

FB インスタンス エリア	先頭アドレ ス	最終アドレ ス	サイズ	変更可能エリア
保持以外 (注 1、3)	H512 (注 2)	H1407 (注 2)	896	I/O、H、W、D、E
保持 (注 1)	H1408 (注 2)	H1535 (注 2)	128	H、D、E
タイマ	T3072	T4095	1024	T
カウンタ	C3072	C4095	1024	C

注 1： 保持以外（非保持領域）、保持（保持領域）に、D（データメモリ）または E（拡張データメモリ）を指定した場合でも、ビットアクセスが可能です。

注 2： H512～1535 は、ファンクションブロック専用保持リレーエリアです。
これらのエリアは、ユーザプログラム上の命令のオペランドには、指定することはできません。また、内部変数の AT 指定にも、指定することはできません。

注 3： H512～1535 は、一種の保持リレーエリアですが、ここで「保持以外」に設定したアドレスは、電源 OFF→ON 時または運転開始時にクリアされます。

・ FQM1 の場合

FB インスタンス エリア	先頭アドレ ス	最終アドレ ス	サイズ	変更可能エリア
保持以外 (注 1)	5000	5999	1000	CIO、WR、DM
保持	なし			
タイマ	T206	T255	50	TIM
カウンタ	C206	C255	50	CNT

注 1： 保持以外（非保持領域）に、D（データメモリ）を指定した場合でも、ビットアクセスが可能です。

・ CP1 シリーズ CPU ユニットのの場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	サイズ	
保持以外	H512	H1407	896	CIO、WR、HR、DM ^{*1}
保持	H1408	H1535	128	HR、DM ^{*1}
タイマ	T3072	T4095	1024	TIM
カウンタ	C3072	C4095	1024	CNT

*1. CP1L-L の DM エリアは以下のようになっています。

アドレス	CP1L-L
D0000～D9999	あり
D10000～D31999	存在しない
D32000～D32767	あり

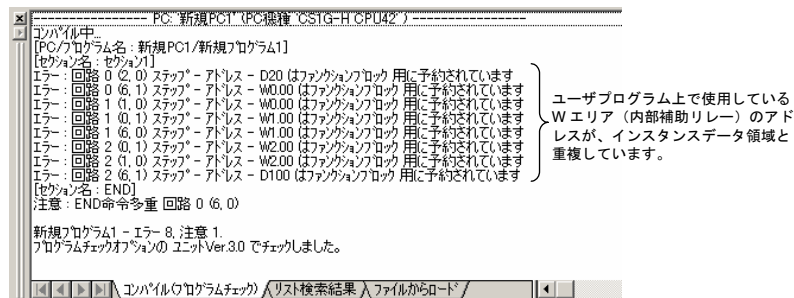
・ CP2E シリーズ CPU ユニットのの場合

FB インスタンス エリア	初期値			割付可能エリア
	先頭アドレス	最終アドレス	サイズ	
保持以外	H512	H1407	896	CIO、WR、HR、DM
保持	H1408	H1535	128	HR、DM
タイマ	T256	T511	256	TIM
カウンタ	C256	C511	256	CNT

お願い

・FB インスタンスエリアとユーザプログラム上での使用アドレスの重複を避けるため、「保持以外」(非保持領域)および「保持」(保持領域)には、基本的に、H512～1535(ファンクションブロック専用保持リレー)を設定してください。
エリアが不足する場合は、ユーザプログラム上で使用していないアドレスを設定してください。

FB インスタンスエリアが、ユーザプログラム上の命令で使用しているアドレスと重複していると、プログラムのコンパイル時(プログラムのダウンロード、オンラインエディット、ユーザによるプログラムチェック実行時)にエラーとなります。
このとき、アウトプットウィンドウの[コンパイル(プログラムチェック)]タブに、以下のようなエラーが表示されます。



重複エラー発生時、FB インスタンスエリアを変更するか、またはユーザプログラム上の命令の使用アドレスを変更してください。

3-2 操作方法

3-2-7 各変数の内部割当アドレスを確認する

3-2-7 各変数の内部割当アドレスを確認する

各変数の、内部的に割り当てられた I/O メモリの実アドレスを確認したい場合、以下の操作を実行します。

- 1 [表示] | [変数テーブル] | [グローバル] を選択します。
- 2 グローバル変数テーブル上のインスタンスを選択し、右クリックして、ポップアップメニューから [ファンクションブロックインスタンス アドレス] を選択します (または、[PLC] メニューから、[メモリ割付] | [ファンクションブロック/SFC メモリ] | [ファンクションブロックインスタンス/SFC アドレス] を選択します)。

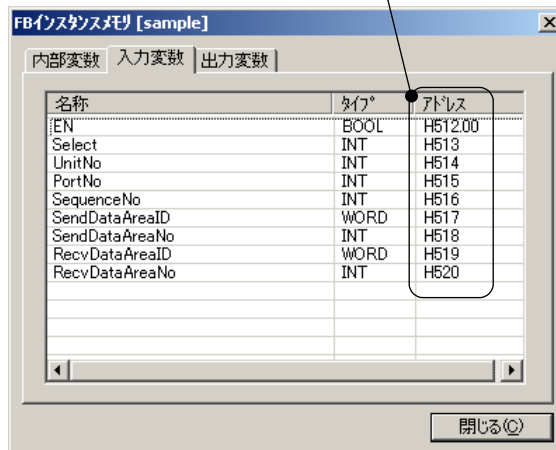
例) グローバル変数上にインスタンス名が表示されます (インスタンスが自動登録されています)

インスタンス名を右クリックして、
[ファンクションブロックインスタンスアドレス]を選択

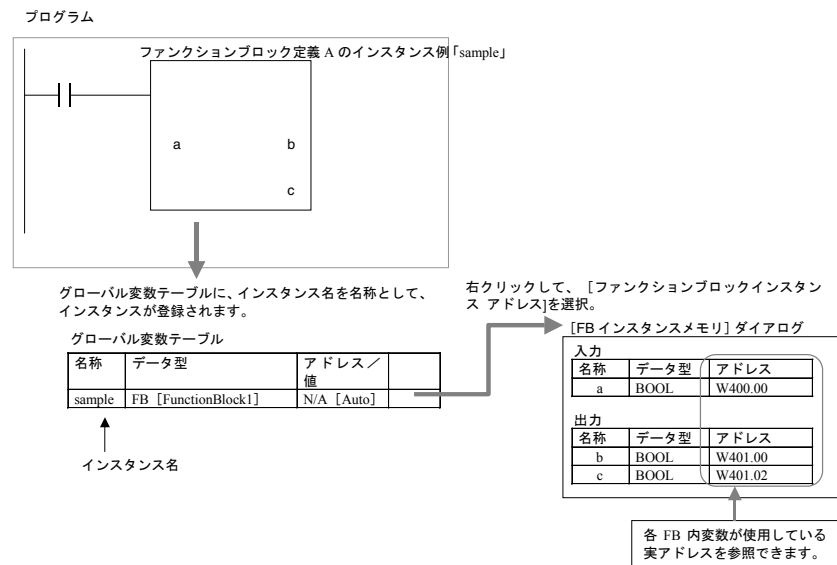


3. [FB インスタンスメモリ] ダイアログボックスが表示され、各変数に内部的に割り当てられている I/O メモリ実アドレスを確認することができます。

例) 入力変数の内部的な使用アドレス



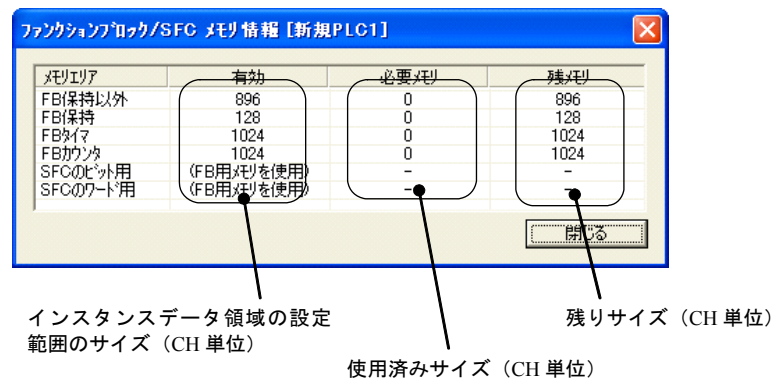
■ 変数の内部割当アドレスを確認するしくみ



■ 変数の内部割当アドレスの使用状況を確認する

変数の内部割当範囲（FB インスタンスエリア）のうち、実際に変数に割り当てられている数と残り数を確認したい場合、以下の操作を実行します。

- 1 ラダーセクションウィンドウ上のインスタンスを選択し、[PLC] メニューから、[メモリ割付] | [ファンクションブロック/SFC メモリ] | [ファンクションブロック/SFC メモリ情報] を選択します。
2. 以下の [ファンクションブロック/SFC メモリ情報] ダイアログボックスが表示されます。



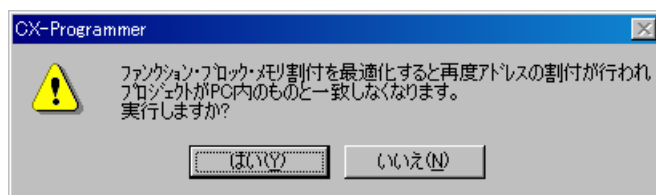
■ 変数の内部割当アドレスの割り当てを最適化する

FB インスタンスエリアへのメモリ割り当てを一度行くと、それ以降は変数の追加または削除を行った時点で、メモリの再割り当てが自動的に行われます。

ただし、1つのインスタンスは連続したアドレスが必要となるため、もし再割り当てで連続したアドレスが確保できないと、すべての変数に別のアドレスを割り当て直します。その結果、未使用領域ができてしまいます。

このような場合、未使用領域をなくし、割り当て領域を効率的に使用するために、以下の操作によって、最適化を実行してください。

- 1 ラダーセクションウィンドウ上のインスタンスを選択し、[PLC] メニューから、[メモリ割付] | [ファンクションブロック/SFC メモリ] | [ファンクションブロック/SFC メモリ割付の最適化]を実行してください。
2. 以下のダイアログボックスが表示されます。



3. [OK] ボタンをクリックすると、FB インスタンスエリアへの割当が最適化されます。

3-2-8 ファンクションブロック定義をコピーし、編集する

作成したファンクションブロック定義をコピーして、編集したい場合は、以下の操作を実行します。

1. コピーしたいファンクションブロックを選択し、右クリックしてポップアップメニューから[コピー]を選択します。
2. プロジェクトツリー上の PLC の下の「ファンクションブロック」アイテムにカーソルを置いて、右クリックしてポップアップメニューから[貼り付け]を選択します。
3. ファンクションブロック定義が、コピーされます（コピー元のファンクションブロック定義名の前に「コピー～」が付きます）。
4. 必要に応じて、左クリック、または右クリックしてポップアップメニューから[名前の変更]を選択してファンクションブロック名を変更できます。
5. ファンクションブロック定義をダブルクリックして、編集できます。

3-2-9 インスタンスから元のファンクションブロック定義を確認表示する

作成したインスタンスの元のファンクションブロック定義の内容を確認したい場合は、以下の操作を実行します。

1. インスタンスにカーソルを置いてダブルクリック、または右クリックしてポップアップメニューから[下の階層へ]を選択します。
2. ファンクションブロック定義が表示されます。

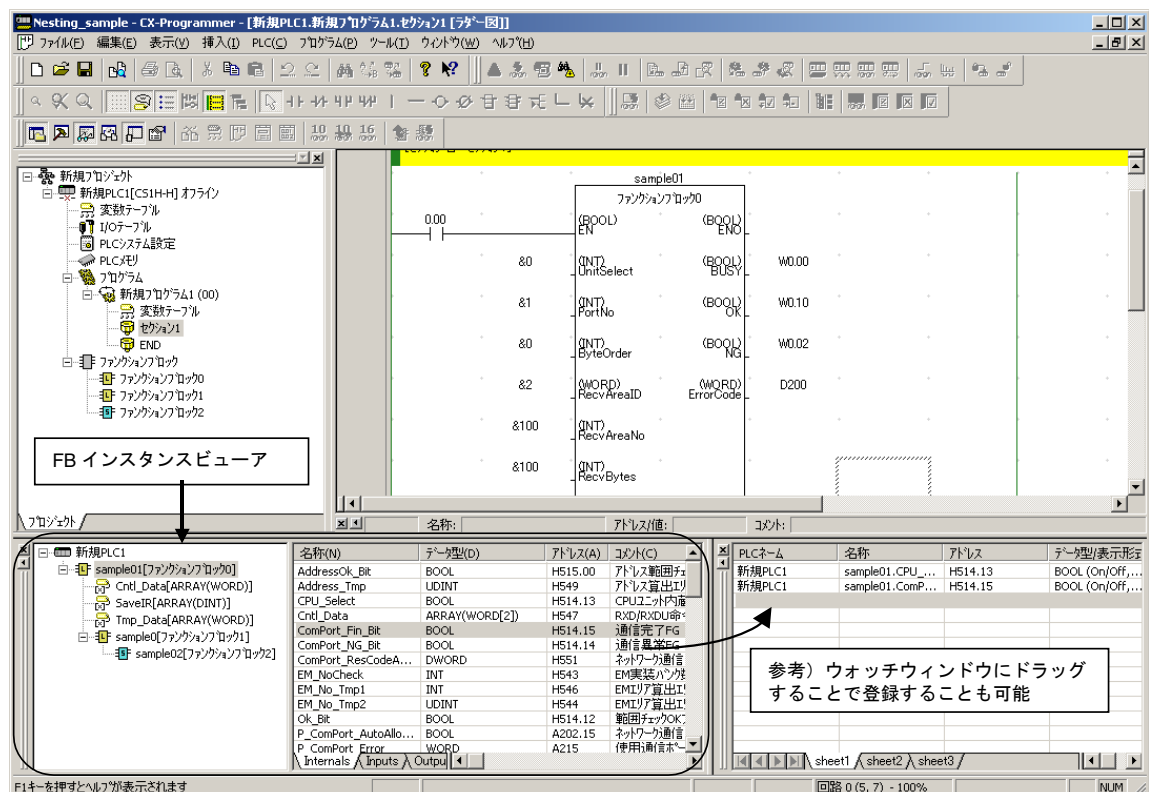
3-2-10 インスタンスの階層構造などの、インスタンスの確認する

作成したプログラム内で、ファンクションブロックをネスティングしている場合、インスタンスの階層構造を確認するとき、[表示]メニューから[ウィンドウ] | [FBインスタンスビュー]を選択します。

呼び出し元が上、呼び出し先が下となったツリーが表示されます。

また、[FBインスタンスビュー]には、インスタンスの階層関係以外に、以下のように、使用している配列変数、使用している変数の内部割当アドレスなども表示されます。

なお、インスタンス内使用変数のリストから、ドラッグ&ドロップで、ウォッチウィンドウへ登録することもできます。



ネスティングしている場合、インスタンス（カッコ内にファンクションブロック定義名）の階層関係が表示されます。
（上が呼び出し元、下が呼び出し先）
なお、上記のように、配列変数、タイマ型/カウンタ型の変数がある場合、インスタンスのすぐ下に表示されます。

左のツリーでアクティブなインスタンス内の使用変数の名称、データ型、アドレス（内部割当アドレス）、コメントが表示されます。

3-2-11 ファンクションブロックの使用量を確認する

ファンクションブロックを使用すると、以下のメモリを消費します。

- ・UM (ユーザメモリ)

ファンクションブロック実行プログラム (オブジェクトコード) が格納されます。

- ・FB ソースメモリ

ファンクションブロック定義 (ファンクションブロックの変数テーブルを含む) を CX-Programmer 上で表示するための FB ソースプログラムが格納されます。

メモリの使用量は以下の手順により確認することができます。

- 1 [表示] メニューから [メモリ表示] を選択します。
- 2 [メモリ表示] ダイアログボックスが、以下のように表示されます。

例：CJ2H-CPU68 の場合

メモリ表示		FBソースメモリ		使用数	
UM		使用量:	967 バイト	使用数:	4
使用量:	630 ステップ*	空容量:	1702457 バイト	未使用数:	2044
空容量:	409994 ステップ*	合計:	1703424 バイト	最大数:	2048
合計:	410624 ステップ*				

メモリ表示ダイアログボックスの表示は、PLC 機種によって異なります。
メモリ表示ダイアログボックスの表示の詳細については、CX-Programmer オペレーションマニュアルの「4-8 メモリ表示機能」を参照してください。

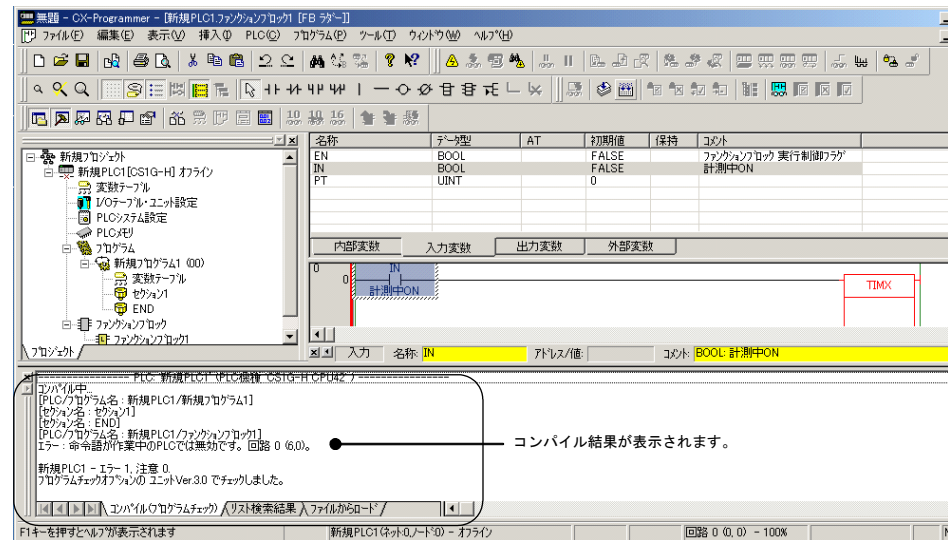
ファンクションブロック実行データのプログラムステップ数の計算方法は、2-9 項「ファンクションブロック使用時の、プログラムステップ数とインスタンス実行時間について」を参照してください。

3-2-12 ファンクションブロック定義をコンパイル (プログラムチェック) する

ファンクションブロック定義だけをコンパイルして、プログラムチェックをかけることができます。以下の操作を実行します。

- 1 ファンクションブロック定義を選択し、右クリックしてポップアップメニューの [コンパイル (プログラムチェック)] を選択、または「Ctrl」+「F7」キーを押します。
- 2 コンパイルが実行され、アウトプットウィンドウの「コンパイル」タブが自動的に表示されて、プログラムチェック結果が表示されます。

3-2-12 ファンクションブロック定義をコンパイル (プログラムチェック) する



3-2 操作方法

3-2-13 ファンクションブロック定義を印刷する

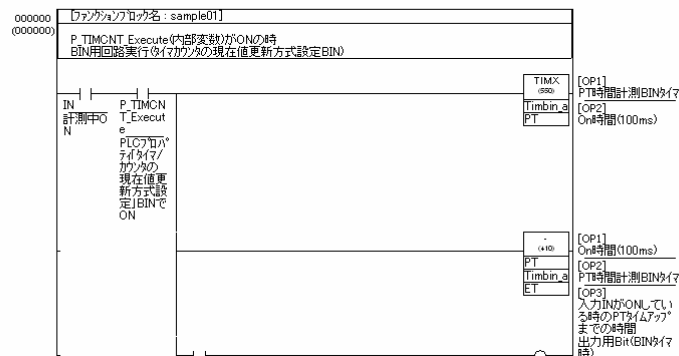
3-2-13 ファンクションブロック定義を印刷する

作成したファンクションブロック定義を印刷することができます。

- 印刷したいファンクションブロック定義をダブルクリックして、変数テーブルとアルゴリズムが表示された状態で、[ファイル]メニューから[印刷]を選択します。以下の[印刷範囲指定]ダイアログボックスが表示されます。

- 印刷範囲として、「すべて」または「回路指定」をチェックします。「回路指定」をチェックしたときは、開始回路番号と終了回路番号を指定します。
[ファイル]メニューの[ページ設定]にて、ヘッダ、フッタのフィールドにページ番号を指定している場合は、先頭のページ番号を指定できます。
- ファンクションブロックの印刷範囲として、以下のいずれかを選択できます。
 - ・変数テーブルとプログラムの印字（デフォルト）
 - ・変数テーブルのみの印字
 - ・プログラムのみの印字
- [OK] ボタンをクリックすると、[印刷] ダイアログボックスが表示されます。プリンタ、印刷部数、用紙の設定を行った後、[OK] ボタンをクリックします。
- 以下のように、変数テーブルと、その後に、アルゴリズム（例：ラダー言語）が印刷されます。

変数タイプ	変数名	データ型	保持	アドレス	初期値	コメント
入力ラミナ	EN	BOOL	なし		FALSE	ファンクションブロック実行制動フラグ
入力ラミナ	IN	BOOL	なし		FALSE	計測中ON
入力ラミナ	PT	UINT	なし		0	On時間(100ms)
出力ラミナ	ENO	BOOL	なし		FALSE	ファンクションブロック実行結果フラグ
出力ラミナ	ET	UINT	なし		0	入力INがONしている時のPTがタイムアウトまでの時間
内部変数	P_TIMCNT_Execute	BOOL	なし	A99.15		PLCのタイマ/カウンタの現在値更新方式設定: BINでON
内部変数	Timbin_a	TIMER	なし			PT時間計測BINタイマ
内部変数	Timbcd_b	TIMER	なし			PT時間計測BCDタイマ
内部変数	PT_Time_BCD	WORD	なし		0	PT時間(BCD値)
内部変数	On_Bit_BIN	BOOL	なし		FALSE	出力用Bit(BINタイマ時)
内部変数	On_Bit	BOOL	なし		FALSE	BCD値範囲チェックOKフラグ
内部変数	On_Bit_BCD	BOOL	なし		FALSE	出力用Bit(BCDタイマ時)



注：印刷設定については、CX-Programmer オペレーションマニュアル (No.SBCA-337) の 4-17 項「印刷」を参照してください。

3-2-14 ファンクションブロック定義にパスワードプロテクトをかける

■ 概要

プロジェクト上のファンクションブロック定義にパスワードを設定することで、プロテクトを行うことができます。

目的に応じて、以下の2種類のパスワードプロテクトを選択できます。

● 書込、表示とも禁止にするパスワードプロテクト

ファンクションブロック定義の内部を、書込（変更）も表示も、ともにできないようにすることが可能です。

ファンクションブロックのプロパティにて、プロテクト種別として「書込/表示禁止」を選択して、パスワードプロテクトを設定します。

これにより、プログラム上のノウハウの流出およびプログラムの不用意な変更、改ざんを防止することができます。

● 書込のみ禁止にするパスワードプロテクト

ファンクションブロック定義の内部を書込（変更）できないようにすることが可能です。表示は可能です。

ファンクションブロックのプロパティにて、プロテクト種別として「書込のみ禁止」を選択して、パスワードプロテクトを設定します。

これにより、プログラム上のノウハウの流出を防止することができます。

■ パスワードプロテクトを設定する

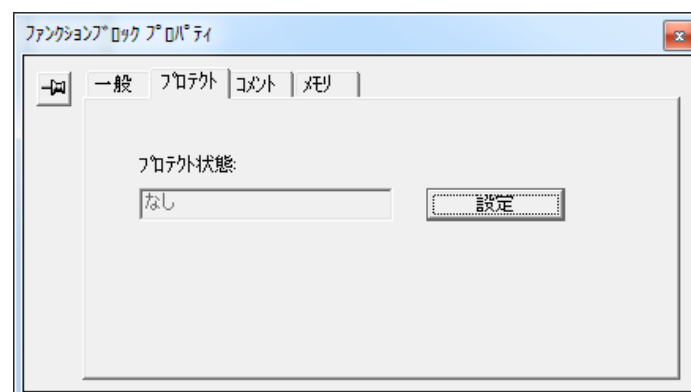
この操作はオフラインでのみ行うことができます。

ファンクションブロック定義ごとに個別で設定することも、複数のファンクションブロック定義に一括で設定することも可能です。

● ファンクションブロック定義ごとに個別で設定する場合

以下の手順で設定します。

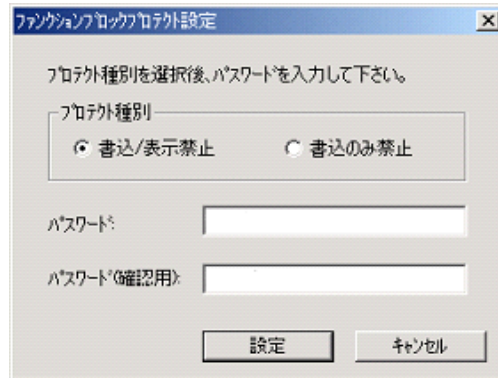
1. プロジェクトツリーで、プロテクトをかけたいファンクションブロック定義をアクティブ表示している状態で、[表示] | [プロパティ] を選択、または右クリックしてポップアップメニューから [プロパティ] を選択します。
2. [ファンクションブロックプロパティ] が表示されますので、[プロテクト] タブを選択し、[設定] を選択します。



3-2 操作方法

3-2-14 ファンクションブロック定義にパスワードプロテクトをかける

3. 以下の「ファンクションブロックプロテクト設定」ダイアログが表示されますので、
「プロテクト種別」フィールドで、プロテクト種別を選択します。



プロテクト種別に応じた機能制限は、以下のとおりです。

×：不可、○：可能

機能	プロテクト種別	
	「書込／表示禁止」の場合	「書込のみ禁止」の場合
ファンクションブロック内部の表示	×	○
ファンクションブロック内部の印刷		×
ファンクションブロック内部の編集		
ファンクションブロックライブラリファイルへのセーブ/ロード	○	○

4. 「ファンクションブロックプロテクト設定」ダイアログの「パスワード」フィールドに、
パスワードを入力します。
確認のために「パスワード（確認用）」フィールドに同じパスワードを入力し、「設定」
ボタンをクリックします。
パスワードは 8 文字以下の半角英数字のみ有効です。

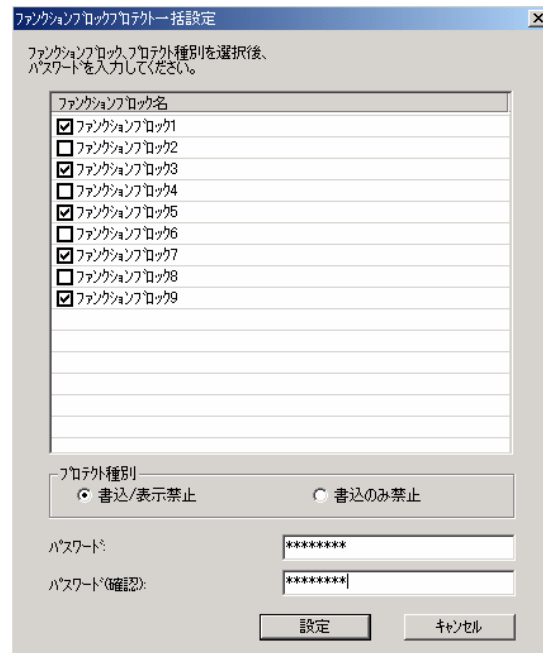
5. 以下の例のようにファンクションブロック定義のアイコンが変化し、設定されたプロテクト種別に沿ったプロテクトが設定されます。

- ：書き込み/表示禁止（ラダー／ST 共通）
- ：書き込みのみ禁止（ラダー）
- ：書き込みのみ禁止（ST）

● プロジェクトファイルの複数のファンクションブロック定義に一括で設定する場合

以下の手順で設定します。

- プロジェクトツリーで、ファンクションブロックアイコンを選択している状態で、
右クリックしてポップアップメニューから「ファンクションブロックプロテクト」
| 「設定」を選択します。
- 以下の「ファンクションブロックプロテクト一括設定」ダイアログが表示されます
ので、プロテクトをかけたいファンクションブロック名をチェック、プロテクト種
別の選択、およびパスワードを入力した後、「設定」ボタンをクリックします。



3. 設定された内容にもとづき、パスワードプロテクトが設定されます。

■ パスワードプロテクトを解除する

この操作はオフラインでのみ行うことができます。

ファンクションブロック定義ごとに個別で解除することも、複数のファンクションブロック定義に一括で解除することも可能です。

● ファンクションブロック定義ごとに個別で解除する場合

以下の手順で設定します。

1. プロジェクトツリーで、プロテクトを解除したいファンクションブロック定義をアクティブ表示している状態で、[表示] | [プロパティ] を選択、または右クリックしてポップアップメニューから [プロパティ] を選択します。
2. [ファンクションブロックプロパティ] が表示されますので、[プロテクト] タブを選択し、[解除] ボタンをクリックします。
3. [ファンクションブロックプロテクト解除]ダイアログが表示されますので、[パスワード]フィールドでパスワードを入力した後、[解除] ボタンをクリックします。
4. パスワードが一致している場合、プロテクトが解除され、プロジェクトツリーのアイコンが通常のアイコンに変化します。

● プロジェクトファイルの複数のファンクションブロック定義に一括で解除する場合

以下の手順で設定します。

1. プロジェクトツリーで、ファンクションブロックアイコンを選択している状態で、右クリックしてポップアップメニューから [ファンクションブロックプロテクト] | [解除] を選択します。
2. [ファンクションブロックプロテクト一括解除] ダイアログが表示されますので、プロテクトを解除したいファンクションブロック名をチェック、パスワードを入力した後、[解除] ボタンをクリックします。
3. 選択されたファンクションブロックのパスワードと入力されたパスワードが全て一致している場合に一括でプロテクトが解除されます。

3-2-15 ファンクションブロックの照合

作業中のファンクションブロックと実機 PLC または他のプロジェクトファイル間で、ファンクションブロック同じかどうかの照合ができます。

プログラムの照合についての詳細は、CX-Programmer オペレーションマニュアル (SBCA-337) を参照してください。

3-2-16 ファンクションブロック定義をファイルに保存して再利用する

作成したファンクションブロック定義だけをファイル（「ファンクションブロック・ライブラリファイル」拡張子.cxf）に保存し、他のプロジェクトで再利用することができます。

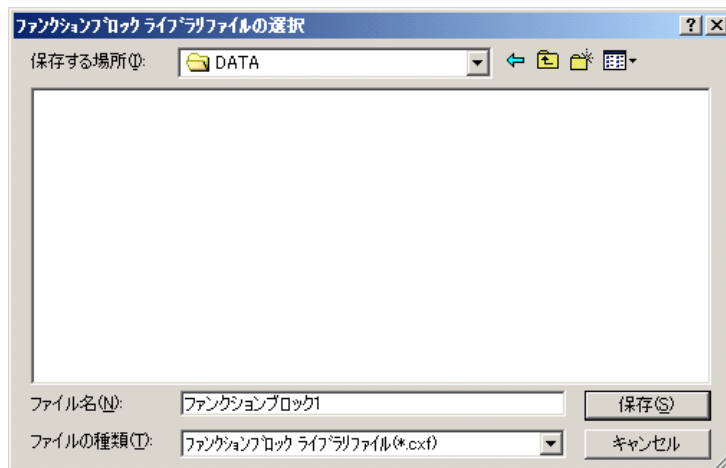
注1：ファイルに保存する前、または他のプロジェクトで再利用するときに、コンパイルを実行して、プログラムチェックをかけてください。

注2：ネスティングしているファンクションブロック定義の場合、呼び出し先のファンクションブロック定義を含めて、ファンクションブロック・ライブラリファイルとして保存されます。

■ ファンクションブロック・ライブラリファイルを保存する

ファンクションブロック・ライブラリファイルを作成して、保存する場合は、以下の操作を実行します。

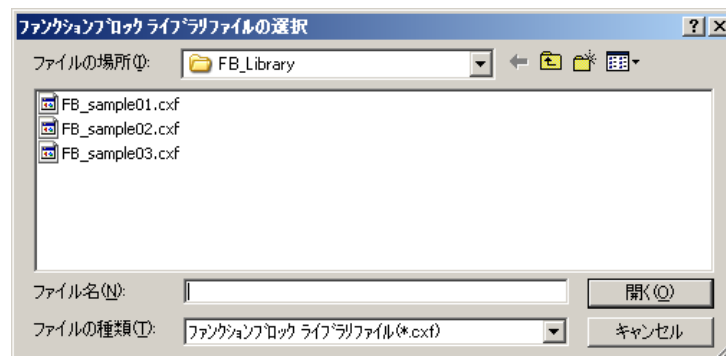
1. 保存したいファンクションブロック定義を選択し、右クリックしてポップアップメニューの「ファンクションブロックをファイルへ保存」を選択します（または「ファイル」メニューから「ファンクションブロック」 | 「ファンクションブロックをファイルへ保存」を選択します）。
2. 以下の保存ダイアログが表示されますので、ファイル名を入力して保存します。ファイルの種類は、「ファンクションブロックライブラリファイル (*.cxf)」を選択します。



■ ファンクションブロック・ライブラリファイルを他のプロジェクトで読み出す

保存したファンクションブロック・ライブラリファイル（拡張子.cxf）を読み出す場合は、以下の操作を実行します。

1. プロジェクトワークスペースのツリー上の PLC の下のファンクションブロックアイテムを選択し、右クリックしてポップアップメニューから「ファンクションブロック挿入」 | 「ライブラリファイル」を選択します（または「ファイル」メニューから「ファンクションブロック」 | 「ファンクションブロックをファイルからロード」を選択します）。
2. 以下のダイアログが表示されます。保存してあるファンクションブロック・ライブラリファイル（拡張子.cxf）を指定して「開く」ボタンをクリックします。



3. 「ファンクションブロック」アイテムの下に、アイコンの後に「ファンクションブロック 1」が自動的に挿入されます。
4. 挿入された「ファンクションブロック 1」をダブルクリックすると、その内容（「変数のテーブル」と「アルゴリズム」）が表示されます。

3-2-17 プログラムを実機 PLC へダウンロード／アップロードする

ファンクションブロックを使用したプログラムを作成後、CX-Programmer を実機 PLC とオンライン接続し、実機 PLC へプログラムをダウンロードします。また、逆にプログラムを実機 PLC からアップロードします。

なお、CX-Programmer Ver.5.0 以降（パソコン）と実機 PLC のプログラムが一致しているかどうかを確認することも可能です。

ただし、ファンクションブロックを含んでいると、タスク単位でダウンロードすることはできません（タスク単位のアップロードは可能です）。

3-2-18 ファンクションブロックをモニタ/デバッグする

ファンクションブロックを使用した回路を、以下の方法で、モニタリングすることができます。

■ インスタンス内ラダープログラムの導通モニタ

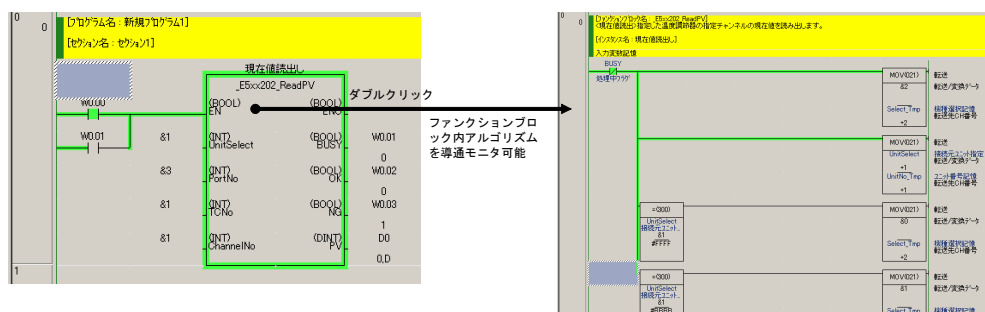
CX-Programmer Ver.6.0 以降では、プログラムをモニタ中に、インスタンスの内部のラダー図を導通モニタすることができます。

インスタンスをダブルクリックするか、インスタンスを右クリックし [FB モニタ] を選択することで、可能です。

このとき、導通モニタ、現在値変更、強制セット/リセット、微分モニタが可能です。

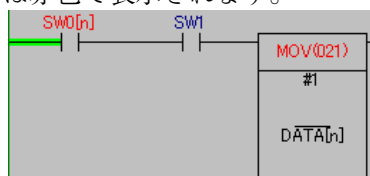
注1： タイマ/カウンタの設定値変更はできません。

注2： 入出力変数は、現在値変更、強制セット/リセットを行うことはできません。かつ、入出力変数として構造体を使用する場合、その構造体の BOOL 型メンバの強制セット/リセット情報「鍵マーク」は、表示できません。



注3： ファンクションブロック内で配列変数を使用している場合

配列変数の引数が変数の場合に、その配列変数を接点や応用命令のオペランドに使用すると、現在値はモニタリングできません。このとき該当する接点、命令は赤色で表示されます。

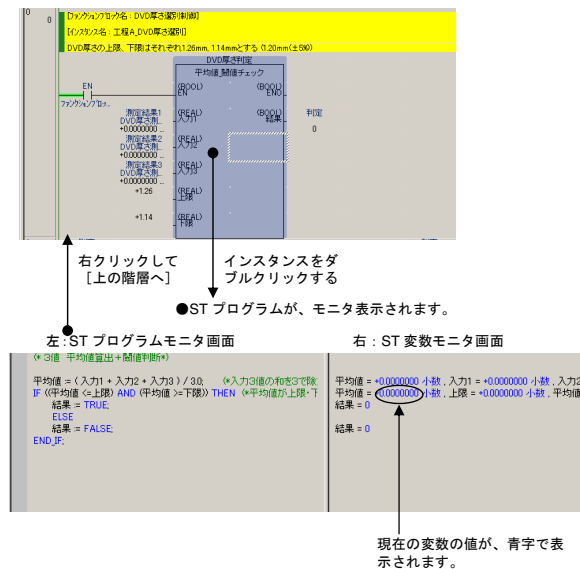


■ インスタンス内 ST プログラムの変数モニタ

CX-Programmer Ver.6.1 以降では、プログラムをモニタ中に、インスタンスの内部の ST プログラムをモニタすることができます。

インスタンスをダブルクリックするか、またはインスタンスを右クリックしてポップアップメニューから [FB モニタ] を選択することで可能です。

元のインスタンスに戻るには、左の ST プログラムモニタ画面にて、右クリックしてポップアップメニューから [上の階層へ] を選択します。



左側（「ST プログラム画面」と呼びます）に、ST プログラムが表示されます。

右側（「ST 変数モニタ画面」と呼びます）に、ST で使用している変数の値が表示されます。

このとき、以下のように、変数モニタ、現在値変更、強制セット/リセット、ウォッチウィンドウへのコピー/ペーストが可能です。

・変数モニタ

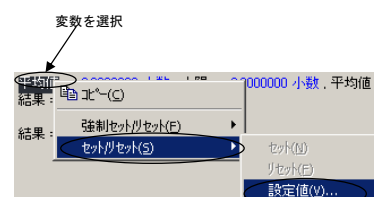
ST 変数モニタ画面内の変数の値が青字で表示されます。

注 1：CJ2CPU ユニットにおいて、0.1ms タイマ命令および 0.01ms タイマ命令の引数の TIMER 型変数については現在値を取得することができないため、ST モニタビューでは現在値は“-“と表示されます。

注 2：CJ2CPU ユニットにおいて、0.1ms タイマ命令および 0.01ms タイマ命令の引数の TIMER 型変数の現在値を、タイマ命令以外の箇所で使用している場合、もしくはその現在値を受けた別変数では、現在値を正しく表示できません。（不定な値が表示されます）

・現在値変更

ST 変数モニタ画面内の変数自体を選択して（反転させて）、右クリックしてポップアップメニューから [セット/リセット] | [設定値] を選択します。



[新しい値を設定] ダイアログボックスが表示されます。

[値] を入力します。

・強制セット/リセット

ST 変数モニタ画面内の変数自体を選択して（反転させて）、右クリックしてポップアップメニューから [強制セット/リセット] | [セット] / [リセット] / [キャンセル] / [すべて解除] を選択します。

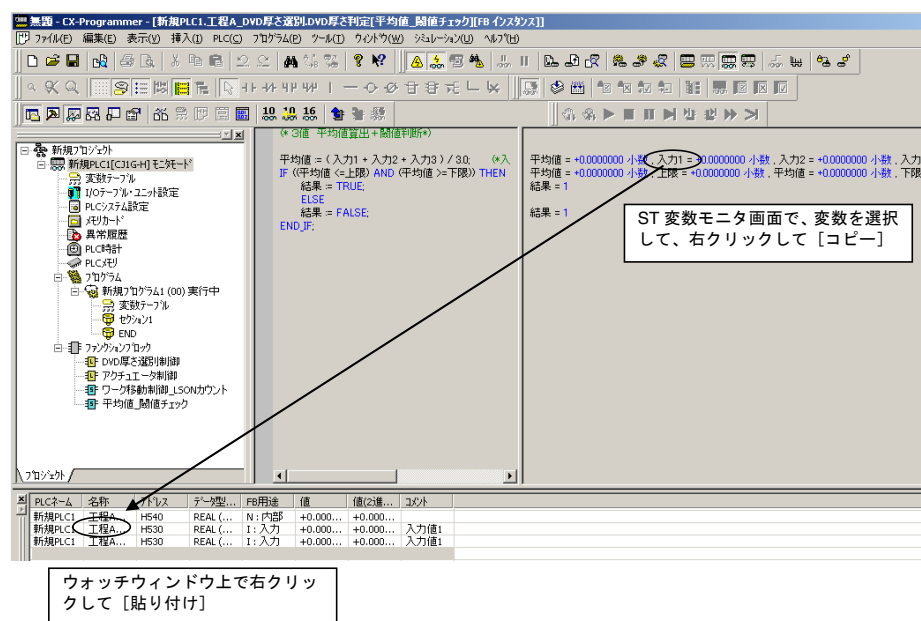
・ウォッチウィンドウへのコピー/ペースト

1. ST 変数モニタ画面内の変数自体を選択して（反転させて）、右クリックしてポップアップメニューから [コピー] を選択します。

3-2 操作方法

3-2-18 ファンクションブロックをモニタ/デバッグする

2. ウォッチウィンドウ内にて右クリックし「貼り付け」を選択します。



■ ファンクションブロック定義のプログラム内容を確認する

モニタ中のインスタンスの元のファンクションブロック定義のプログラムを確認したい場合は、以下の操作を実行します。

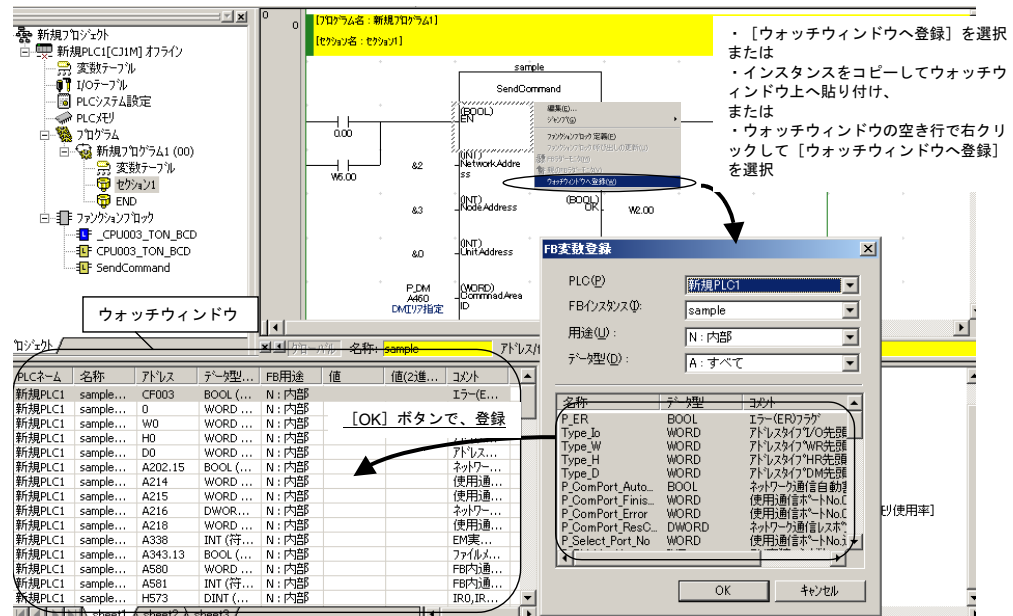
1. インスタンスにカーソルを置いて、右クリックしてポップアップメニューから「ジャンプ」 | 「ファンクションブロック定義」を選択します。
2. ファンクションブロック定義が表示されます。

■ インスタンスの変数をウォッチウィンドウでモニタする

インスタンスの変数をモニタする場合は、以下の操作を実行します。

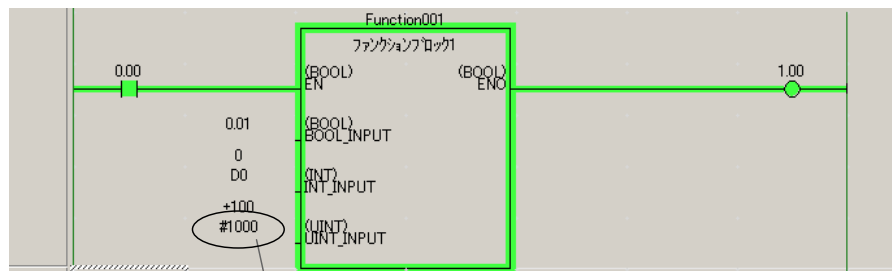
1. 「表示」 | 「ウィンドウ」 | 「ウォッチウィンドウ」を選択します。
2. ウォッチウィンドウが表示されます。
3. 以下の3つのいずれか方法によって、「FB 変数登録」ダイアログボックスを表示します。
 - ① インスタンスを右クリックしてポップアップメニューから「ウォッチウィンドウへ登録」を選択する。または、
 - ② インスタンスをコピーしてウォッチウィンドウ上へ貼り付ける。または、
 - ③ ウォッチウィンドウの空き行で右クリックして「ウォッチウィンドウへ登録」を選択する
4. 「用途」、「データ型」を選択します。その他「FB インスタンス」も選択できます。「用途」：デフォルトでは「N：内部」が登録されます。他に「I：入力」、「O：出力」、「E：外部」から選択できます。
「データ型」：デフォルトでは「A：すべて」が登録されます。「BOOL」「INT」などの特定のデータ型を選択できます。

5. [OK] ボタンをクリックします。ウォッチウィンドウに、以下のように、ウォッチウィンドウの対象とする変数が登録され、値が表示されます。



■ インスタンスの入力変数/出力変数/入出力変数の値のモニタ

インスタンスの入力変数/出力変数の値（パラメータの現在値）は、パラメータの下に表示されます。



入力変数、出力変数の受け渡し元パラメータの現在値を表示します。

■ インスタンス内のラダー/ST プログラムのシミュレーション機能

CX-One Ver.1.1 (CX-Programmer Ver6.1) 以降では、ファンクションブロックのインスタンス内のラダープログラムおよび ST プログラムのシミュレーション機能が可能です。

ステップ実行およびブレークポイントが可能です。

● シミュレーション機能を有効とするまでの操作

以下の操作により、シミュレーション機能を有効にします。

1. デバックしたい、インスタンスを含むプログラムを開きます。
2. [表示]メニューの[ツールバー]を選択し、[ツールバー]タブの[シミュレータデバッグ]をチェックします。
3. CX-Programmer の [PLC] メニューから [シミュレータオンライン接続] を選択して、パソコン内の CX-Simulator へプログラムを転送します。

注：手順 2.と 3.は、逆でもかまいません。

3-2 操作方法

3-2-18 ファンクションブロックをモニタ/デバッグする

● ステップ実行

ステップ（命令）単位の実行を行います。

インスタンスで停止した状態から、そのインスタンス内のラダー/ST プログラムの先頭ステップ（命令）に移行できます。

インスタンス内で、ステップ実行または連続ステップ実行（注）が可能です。

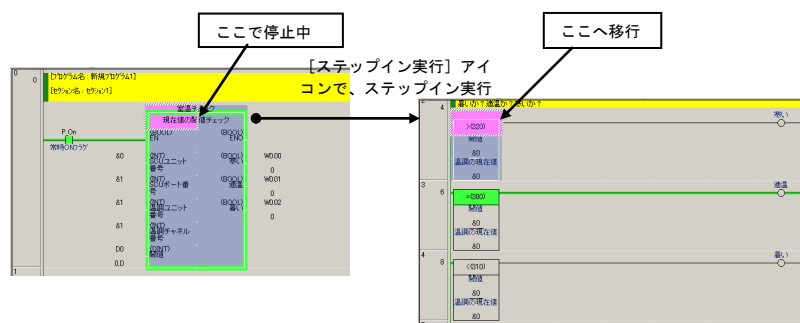
注：連続ステップ実行のステップ実行のインターバルは、CX-Programmer の [ツール] | [オプション] の「連続ステップ実行間隔」で設定します。

・ ステップイン実行

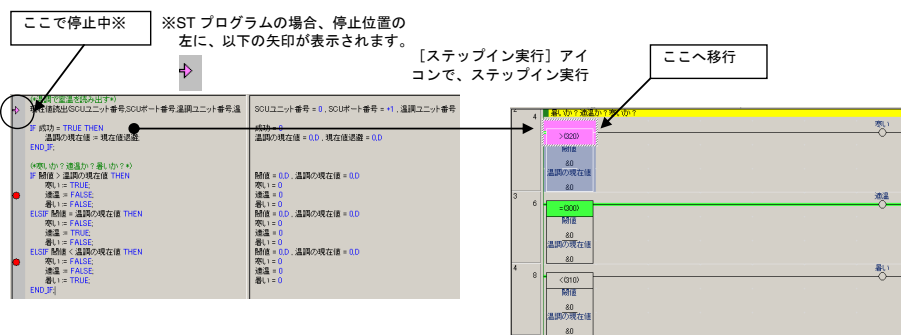
インスタンス内部のラダー/ST プログラムのステップ実行に入る場合（これを「ステップイン実行」と呼びます）、以下の操作を行います。

1. インスタンス上で一時停止状態にします（注）。
2. [ステップイン実行] アイコンをクリックまたは [ツール] メニューから [シミュレーション] | [モード] | [ステップイン実行] を選択します。

・ インスタンスから、内部ラダープログラムへのステップイン実行の例



・ ST プログラムから、さらに内部ラダープログラムへのステップイン実行の例



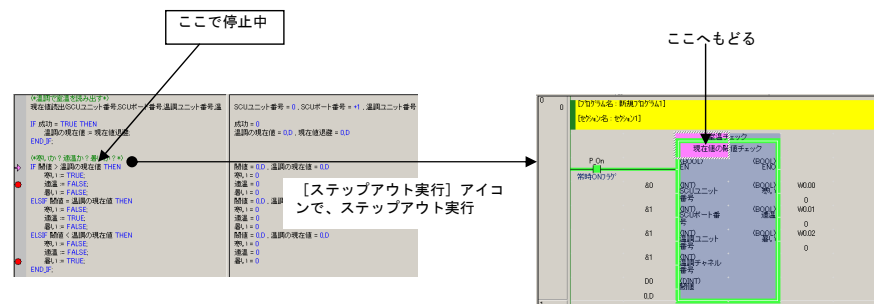
注：実行位置がファンクションブロックインスタンス以外の場合は、通常のステップ実行と同じ処理になります。

・ ステップアウト実行

インスタンスの内部のラダー/ST プログラムのステップ実行中に、1つ上の階層（呼び出し元プログラムまたはインスタンス）に戻って一時停止する場合（これを「ステップアウト実行」と呼ぶ）、以下の操作を行います。

1. ステップ実行中に、インスタンスの内部のいずれかの停止位置にて、
2. [ステップアウト実行] アイコンをクリック、または [ツール] メニューから [シミュレーション] | [モード] | [ステップアウト実行] を選択します。

- ・インスタンス内部 ST プログラムから、1つ上の階層（呼び出し元プログラムまたはインスタンス）へのステップアウト実行の例



注：ステップアウト実行は、インスタンスの内部のラダー/ST プログラムでのみ実行できます。

● シミュレーション機能による一時停止時の表示

シミュレーション機能画面では、どの動作による一時停止時かを、カーソル色（ST プログラム上の場合は矢印の色）で区別できます。

シミュレーション機能による一時停止状態のカーソル表示色（または矢印表示色）は、以下のとおりです。

デバッグ動作種類	色(デフォルト)	プログラム実行状態名	内容
ステップ実行、連続ステップ実行	ピンク	シミュレーター一時停止状態	ステップ実行による一時停止または一時停止ボタンによる一時停止の状態です。
	通常の色	インターロック等の非実行状態	IL、MILR/MILH、JMP0、FOR/BREAK 等で非実行状態でステップ実行している状態です。
ブレークポイント	青	シミュレータ命令ブレーク	ブレークポイントによる一時停止（ブレーク）状態です

参考

- ・[ツール] メニューから[シミュレーション] | [現在の実行位置を常に表示] を選択すると、ステップ実行または連続ステップ実行中に、インスタンス内での一時停止位置を、ステップ実行または連続ステップ実行に合わせて、自動的にスクロール表示することができます。
- ・シミュレーション機能による一時停止状態のカーソル表示色（または ST プログラムのときは矢印表示色）をデフォルト状態から変更することができます。
[ツール] | [オプション] の[表示] タブにて、項目として、「シミュレーター一時停止状態」、「シミュレータ命令ブレーク」、「シミュレータ IO ブレーク設定」を選択し、その色を変更できます。

● インスタンス内のブレークポイント動作

インスタンス内のブレークポイントを指定した位置で、自動的に一時停止させることができます（この場合、ステップイン実行の操作は不要です）。

注：あるインスタンス内で設定されたブレークポイントは、そのインスタンス内だけで有効です（同じファンクションブロック定義を使用した他のインスタンスでは有効になりません）。

3-2-19 ファンクションブロック定義のオンラインエディット

PLC（CPU ユニット）が運転中の場合でも、ファンクションブロック定義（ラダー、ST）の内容を変更することができます。

これにより、24 時間稼働など、運転を停止できない装置においても、ファンクションブロック定義自体のデバッグ/変更が可能となります。

ただし、CX-Programmer Ver.7.0（CX-One Ver.2.0）以降で、かつ CS/CJ シリーズ CPU ユニット Ver.4.0 以降（注）、または CJ2 シリーズ CPU ユニットであることが必要です。

ただし、本機能は、CX-Simulator 上で、シミュレーションで使用することはできません。

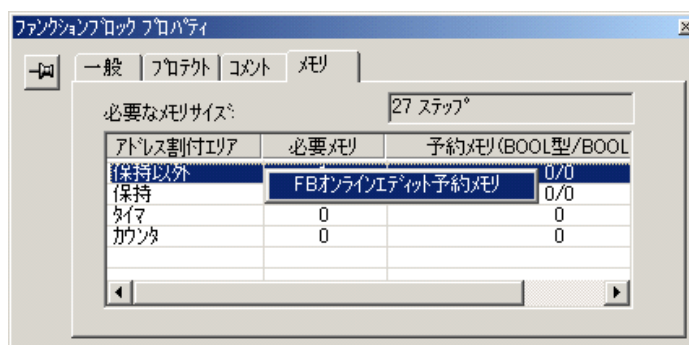
注：CS/CJ シリーズ CPU ユニット Ver.3.0 の場合、ファンクションブロックを使用した回路を、オンラインエディットで変更する場合、インスタンスの周辺を変更することができます。

- ・インスタンスの受け渡し用パラメータの変更、インスタンス以外の命令変更、インスタンスの削除は可能です。
- ・インスタンスの追加、インスタンス名の変更、およびファンクションブロック定義の変数テーブルおよびアルゴリズムの変更は、できません。

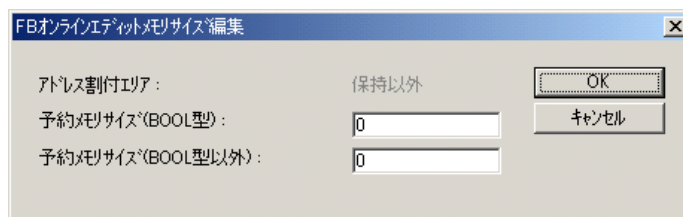
■ オンラインエディットにて内部変数を追加する場合（予約メモリの確保方法）

オンラインエディットにて、ファンクションブロック定義の変数テーブルに内部変数を追加する場合、その追加分の変数サイズを、予約メモリとして、（変数の内部割当範囲（FB インスタンスエリア）とは別に）あらかじめ確保しておく必要があります。オンラインエディット（ファンクションブロック定義の内容の変更）操作の前に、以下の操作によって、予約メモリを確保してください。

1. オンラインエディットの対象となるワークスペース上のファンクションブロック定義を右クリックし、ポップアップメニューから[プロパティ]を選択します。
2. [メモリ]タブを選択します。
追加したいエリアを右クリックして [FB オンラインエディット予約メモリ] を選択します。



2. 以下の[FB オンラインエディットメモリサイズ編集]ダイアログボックスにて、各アドレス割付エリアに予約メモリを入力することができます。

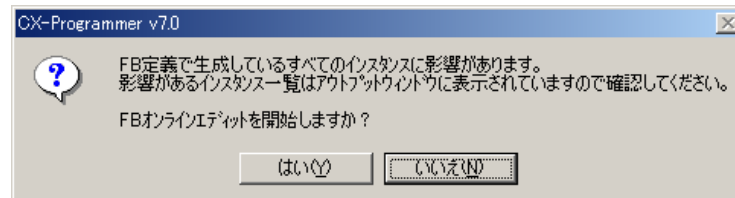


■ ファンクションブロック定義の内容の変更と転送

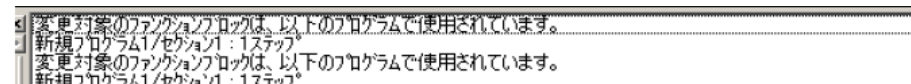
1. PLC とオンライン接続している状態で、ワークスペース上のファンクションブロック定義を選択*1 し、ポップアップメニューから[FB オンラインエディット] | [開始]を選択します。

*1：ファンクションブロック定義画面、インスタンス内ラダー/ST モニタ画面、ファンクションブロック呼び出し命令（ラダープログラム、ファンクションブロック内ラダー言語とも）からも、オンラインエディットが可能です。

2. FB オンラインエディットを開始する前に、以下の確認ダイアログが表示されます。

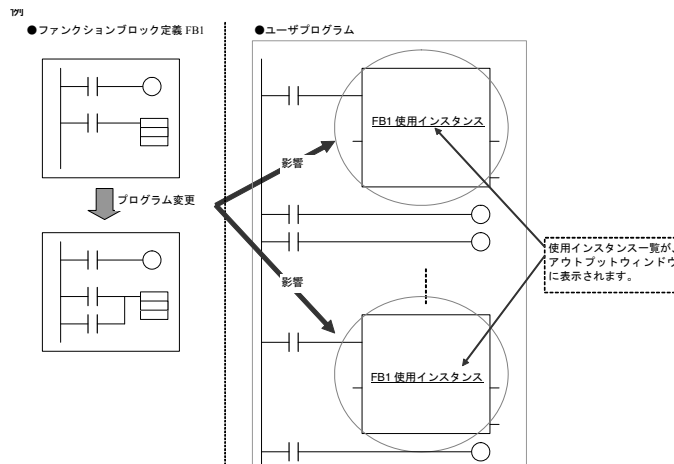


同時に、影響があるインスタンス一覧が、アウトプットウィンドウに、以下のように表示されます。



お願い

ファンクションブロック定義変更のインスタンスへの影響について：ファンクションブロック定義を変更すると、それを使用している（呼び出している）すべてのインスタンスの内容が、同時に変更されます（下図参照）。

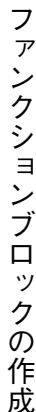


したがって、必ずユーザプログラム全体動作への影響を考慮した上で、ファンクションブロック定義の変更内容をプログラムに反映してください。

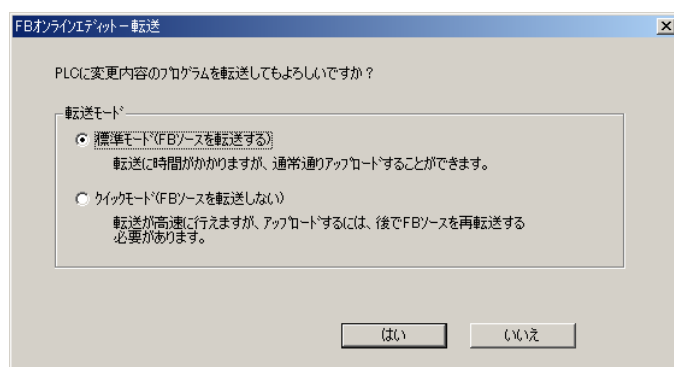
3. [はい] ボタンをクリックすると、選択されたファンクションブロック定義の内容が表示され、内容が編集可能状態になります。

3 ファンクションブロックの作成

ファンクションブロックの作成

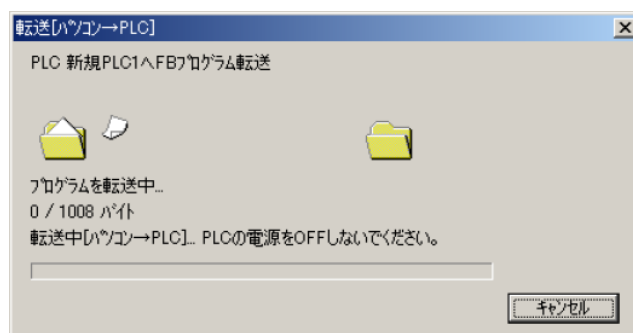


4. 編集後、[FB オンラインエディット] | [変更を転送]を選択します。
以下の「変更を転送」ダイアログボックスが表示されます。



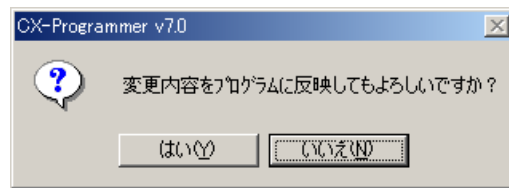
5. 以下のいずれかの転送モードを選択し、[はい] ボタンをクリックします。
- ・標準モード
 - ・クイックモード
- 転送モードの詳細は、後述の「●転送モードについて」、「●転送モードの使い分けについて」を参照してください。

6. 変更したファンクションブロック定義が、PLC（CPU ユニット）のバッファメモリへ転送開始されます。
- 転送の進行状況を示すダイアログボックスが以下のように表示されます。



(この時点では、PLC はまだオンラインエディット前の状態で動作しています。)

7. 転送終了後、以下の確認ダイアログボックスが表示されます。



(この時点では、PLC はまだオンラインエディット前の状態で動作しています。)

8. [はい] ボタンをクリックすると、PLC (CPU ユニット) のバッファメモリにあったファンクションブロック定義のプログラム変更内容が、PLC (CPU ユニット) のプログラムに反映されます。
[いいえ] ボタンをクリックすると、PLC (CPU ユニット) のバッファメモリは破棄され編集中の FB オンラインエディット状態に戻ります。
9. ファンクションブロック定義は編集不可能な状態に戻ります。追加で編集作業を行う場合は、[FB オンラインエディット] | [開始]を選択し、オンラインエディットの開始から始めてください。

● 転送モードについて

・標準モード

ファンクションブロック定義のソースコードとオブジェクトコードをともに PLC (CPU ユニット) に転送します。

「標準モード」では、転送するデータ量により、転送に時間がかかることがあります。その間は次の編集および転送操作を行うことができません。

注: ソースコードを転送後、それを PLC (CPU ユニット) のユーザメモリへ反映する直前に、反映しますかという確認のダイアログボックスが必要な場合、「オンラインエディットの反映確認を表示する」をチェックしてください。

・クイックモード

ファンクションブロック定義のオブジェクトコードのみを PLC (CPU ユニット) に転送します。ソースコードを転送しません。

「クイックモード」では、オブジェクトコードのみを転送するため、「標準モード」より高速で転送が可能です。

「クイックモード」で転送した後は、① [プログラム] メニューから [FB ソース転送] を選択して、または②オフラインに切り替えるときにダイアログボックスの指示にしたがって、ソースコードを転送してください。

変更を転送後、オフライン状態に切り替えるまで、画面下に、「ソースコードが転送されていないこと」を示す黄色の「FB ソース」の文字が表示されます。ソースコード転送終了後、黄色の「FB ソース」の文字が消えます。

● 転送モードの使い分けについて

基本的に、「標準モード」で転送をしてください。転送に時間がかかるときは、できるだけ PLC との通信速度を上げて、転送処理を行ってください。

それでも転送時間がかかりすぎて、デバッグに支障がある場合 (デバッグ時に頻繁に連続してオンラインエディットをするときなど) は、例外的に、以下の「クイックモード (ソースコードの転送なし) の制約」を十分ご理解いただいた上で、「クイックモード」で転送してください。

3-2 操作方法

3-2-19 ファンクションブロック定義のオンラインエディット

転送時間の目安は、以下のとおりです。

例) FB 定義を 8 個使用し、ソースコード (FB 定義 8 個全体+FB インスタンス全体) の容量が 8K バイトのとき

	標準モード	クイックモード
通信速度 115.2kbps のとき：	5 秒	1 秒
通信速度 19.2kbps のとき：	10 秒	2 秒

お願い

クイックモード (ソースコードの転送なし) の制約：

ファンクションブロック定義のソースコードを転送していない状態では、**CX-Programmer** は、ファンクションブロックを含めた正しいプログラムを次にアップロードすることができません。

したがって、オンラインエディット時に「クイックモード」でファンクションブロックの変更内容を転送した後、[プログラム] メニューから **[FB ソース転送]** を選択して、ソースコードを転送してください。ただし、もしその **FB ソース転送** の操作をしない場合でも、オンラインからオフラインに切り替える前には、自動的にソースコードが転送されます。

ただし、ソースコード転送終了前に、万一パソコン (**CX-Programmer**) のダウンが発生した場合は、**CX-Programmer** がプログラムをアップロードできない状態に陥る可能性がありますので、ご注意ください (*1)。

*1：なお、その場合、以下に示す操作によって、ソースコードを再度転送することができる場合があります。

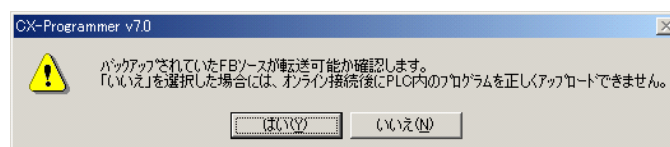
1. **CX-Programmer** を起動します。起動時に、以下のダイアログボックスが表示されます。



2. [OK] ボタンをクリックします。

3. 以前クイック転送を実行した PLC にオンライン接続します。

4. オンライン接続をすると、パソコン内に自動的にバックアップされていた CXP プロジェクトが開かれます。以下のダイアログボックスが表示されます。



5. [はい] ボタンをクリックします。

以降ダイアログボックスの指示にしたがって、操作を行ってください。

パソコン内に自動的にバックアップされていたソースコードと、実機 PLC 内のオブジェクトコードを照合し、一致した場合、ソースコードの転送を行うことができます。

参考

「ソースコード」と「オブジェクトコード」について：

通常、**CX-Programmer** は、プログラム転送前に、大元となるプログラムコード（「ソースコード」）を、**CPU** ユニットが実行可能なコード（「オブジェクトコード」）に変換（コンパイル）して、その両方を転送します。

CPU ユニットは、プログラムのソースコードおよびオブジェクトコードをユーザメモリおよび内蔵フラッシュメモリに保存します。この両方のコードが保存されている場合のみ、**CX-Programmer** は **CPU** ユニットからのアップロード操作によって、転送したプログラムを復元することができます。

3

■ ファンクションブロック定義の内容の変更キャンセル

1. 編集したファンクション定義の内容を破棄する場合は、[FB オンラインエディット] | [取消] を選択します。

編集した FB 定義は、PLC に転送されず、編集前の状態に戻ります。

■ CPU ユニット動作への影響

「モニタ」モードのときに（**CPU** ユニットが運転中に）、オンラインエディットを行うと、

1) **CPU** ユニットのプログラムを書き替えたときに、**CPU** ユニットのサイクルタイムが、複数サイクルにわたって延びます。

また、2) オンラインエディット完了後に、実行結果を内蔵フラッシュメモリにバックアップするため、さらにサイクルタイムが延びます（この間、**CPU** ユニット前面の **BKUP LED** が点灯し、**CX-Programmer** では進行状況が表示されます）。

● オンラインエディットによるサイクルタイムの最大延び時間

オンラインエディットによるサイクルタイムの最大延び時間は、以下のとおりです。

サイクルタイムの最大延び時間	
1) オンラインエディット中	2) バックアップ中
最大 12ms	サイクルタイムの 4%

お願い

サイクルタイム監視時間について：

「モニタ」モードでオンラインエディットによるプログラム書き換え時にサイクルタイムが延び、その結果 PLC システム設定で設定した「サイクルタイム監視時間」を超えることがないようにしてください。もし超えた場合、「サイクルタイムオーバー」エラーとなり、**CPU** ユニットの運転が停止します。再度運転する場合、「プログラム」モードにした後、再び「運転」または「モニタ」モードに戻してください。

⚠ 注意

ユニット間同期制御機能を使用時、オンラインエディットにより同期処理時間が延びると、想定外の動作タイミングとなることがあります。同期処理時間が延びても主軸と従軸の動作に影響がないことを確認してからオンラインエディットをしてください。



■ ファンクションブロックのオンラインエディットにおける制限事項

ファンクションブロックのオンラインエディットにおける制限事項は以下のとおりです。

- ・4K ステップを超えるサイズのファンクションブロック定義は、オンラインエディットできません。（CJ2 シリーズを除く）
- ・1 回のオンラインエディットで変更できる増減サイズは、0.5K ステップです。（CJ2 シリーズを除く）
- ・入力変数、出力変数、入出力変数の追加および削除はできません。
- ・新規のファンクションブロックインスタンスを追加することはできません。
- ・インスタンス名の変更はできません。

・ファンクションブロック定義内の変数テーブルでは、内部変数タブにおいて変数の追加、変数の編集（コメントのみ）、変数の削除が可能です。

ただし、内部変数の追加をする場合、あらかじめそのサイズの予約メモリを確保しておく必要があります。前述の「■ オンラインエディットにて内部変数を追加する場合（予約メモリの確保方法）」を参照してください。

- ・オンラインエディットされたファンクションブロック定義内の立ち上がり微分命令（DIFU/@付命令）/ 立ち下り微分命令（DIFD/%付命令）の前回値のフラグは、オンラインエディット完了後、初期化（OFF）されます。
- ・オンラインエディット後、CPU ユニットが内蔵フラッシュメモリへのバックアップを完了するまで（CPU ユニット前面の BKUP LED が点灯から消灯に変わるまで）、PLC 本体の電源を OFF にしないでください。

バックアップ完了前に（CPU ユニット前面の BKUP LED が点灯中に）PLC 本体の電源を OFF にすると、バックアップが行われず、プログラムはオンラインエディット前の状態に戻ります。

ストラクチャードテキスト編

第4章 ST 言語の概要

4-1 ST 言語とは

本編は、CX-Programmer のタスクに直接割り付ける ST プログラム (ST タスク) の仕様や操作方法を説明しています。

他のプログラム (ファンクションブロック、SFC) で使用される ST プログラム特有の機能や操作については、以下を参照してください。

- ・ ファンクションブロックのインスタンスで使用される ST プログラム
本書のファンクションブロック編
- ・ SFC で使用される ST プログラム
CX-Programmer (SFC 編) オペレーションマニュアル (SBCE-348)

4

ST 言語の概要

4-1-1 概要

ST (Structured Text の略) 言語とは、IEC 61131-3 で規定されている、工業制御 (おもにプログラマブルコントローラ) 用の高級言語です。

用意された制御文・演算子・関数により、ラダーでは記述しにくい数値演算処理を部分的に処理するのに適しています (ラダー言語で記述できる処理すべてを記述することはできません)。

CX-Programmer Ver.7.2 以降では、IEC-61131-3 に準拠した ST 言語を、タスクに割り付けるプログラム言語としてサポートしています。

PLC 機種としては CS/CJ シリーズ CPU ユニット ユニット Ver.4.0 以降、または CJ2 シリーズ CPU ユニットであることが必要です。

以下の特長があります。

- ループ文や IF THEN ELSE などの制御文、四則・比較・AND/OR などの演算子、絶対値や三角関数などの数値演算関数、文字列抽出や結合などの文字列関数、メモリカード処理関数、文字列送受信関数、角度変換関数などを、用意しています。
- C 言語などの高級言語と同様の記述や、漢字を含めた文字でのコメントにより、見やすいプログラムを作成できます。

```
STプログラム：
IF score > setover THEN (* score > setover ならば*)
  underNG := FALSE; (* underNG を OFF *)
  OK := FALSE; (* OK を OFF *)
  overNG := TRUE; (* overNG を ON *)
ELSIF score < setunder THEN (* score <= setover で、score < setunder ならば*)
  overNG := FALSE; (* overNG を ON *)
  OK := FALSE; (* OK を OFF *)
  underNG := TRUE; (* underNG を ON *)
ELSE (* setover > score > setunder ならば*)
  underNG := FALSE; (* underNG を OFF *)
  overNG := FALSE; (* overNG を OFF *)
  OK := TRUE; (* OK を OFF *)
END_IF; (* IF 分の終了*)
```

- ST プログラムは、通常のプログラムと同様、ダウンロード／アップロードできます。
ただし、ST プログラムのタスクを、タスク単位でダウンロード/アップロードすることはできません。
- ST プログラムで、ファンクションブロック (ラダー言語または ST 言語) を呼び出すことができます
- 配列型 (一次元配列) の変数をサポートしているため、アプリケーションに応じたデータハンドリングが容易となります。

4-2 CX-Programmer の仕様

CX-Programmer の ST プログラム (ST タスク) の使用環境を示します。
基本的な CX-Programmer の使用環境は、CX-Programmer オペレーションマニュアル (SBCA-337) を参照してください。
他のプログラム (ファンクションブロック、SFC) で使用される ST プログラムの使用環境については、本書のファンクションブロック編または、CX-Programmer オペレーションマニュアル SFC 編 (No.SBCA-348) を参照してください。

4-2-1 ST プログラム (ST タスク) が使用可能な PLC 機種

以下の PLC (プログラマブルロジックコントローラ) 機種が ST プログラムに対応しています。

PLC 機種	CPU タイプ
CJ2H	CPU68/67/66/65/64/68-EIP/67-EIP/66-EIP/65-EIP/64-EIP
CJ2M	CPU11/12/13/14/15/31/32/33/34/35
CS1G-H ^{*1}	CPU45/44/43/42
CS1H-H ^{*1}	CPU67/66/65/64/63
CS1D-H ^{*1}	CPU68A/67A
CS1D-S ^{*1}	CPU67A/44A
CJ1G-H ^{*1}	CPU45/44/43/42
CJ1H-H ^{*1}	CPU67/66/65/67-R/66-R/65-R/64-R
CJ1M ^{*1}	CPU23/22/21/13/12/11

*1：ユニット Ver.4.0 以降

4-2-2 仕様一覧

項目	内容
タスクに割り付けられるプログラムの言語	SFC、ラダー、ST のいずれか (任意の組み合わせが可能)
ST プログラムの単位	タスク単位 個数：最大 288 (サイクル実行タスク：32、追加タスク：256)
ST プログラムを割り付けることができるタスク	サイクル実行タスクまたは追加タスクに可能
オンラインエディット	ST チャート編集 注：標準モード (ST ソースコードを含んで転送)、クイックモード (ST ソースコードを含まないで転送) を選択可能
配列変数	SFC、ラダー、ST のプログラム内で配列型の変数を使用可能

4-2 CX-Programmer の仕様
4-2-2 仕様一覧

4

S
T
言
語
の
概
要

ストラクチャードテキスト編

第5章 ST 言語の仕様

5-1 ストラクチャードテキスト (ST) 言語の仕様

5-1-1 ストラクチャードテキスト (ST) 言語とは

ST 言語は、構造化テキスト言語です。選択、繰り返しなどの言語構造をもつ、PASCAL に似た高級言語です。

■ ST 言語の構成について

・ ST 言語の構成

ST 言語でのプログラムは、文から構成されます。

文には、以下の 2 種類があります。

- ・ 代入文：式による演算結果を変数に格納する文です。
- ・ 制御文：選択文、繰り返し文などがあります。

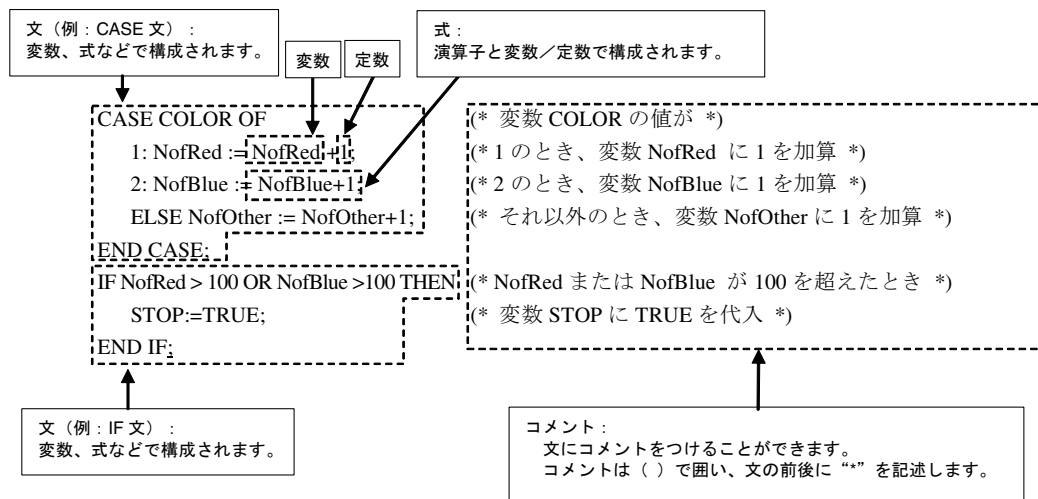
各構文の詳細については、5-5 項を参照してください。

・ 文の構成内容

文は、以下の要素で構成されています。

- ・ 変数 (5-4-2 項参照)
- ・ 定数 (5-4-4 項参照)
- ・ 演算子 (5-4-7 項参照)
- ・ 関数 (5-4-5 項、5-4-6 項参照)

・ 記述例 (制御文の場合)



参 考

ST プログラムでは、アドレスを、I/O メモリの実アドレスでは記述せず、すべて変数名で記述します。

変数が使用するアドレスは、ユーザがそれぞれ設定します。

変数についての仕様や設定方法については、CX-Programmer オペレーションマニュアル (No.SBCA-337) を参照してください。

5-2 ST プログラムで使用するデータ型

ST プログラムで使用するデータ型を以下に示します。

ファンクションブロック内で ST を使用する場合に使用可能なデータ型は、ファンクションブロック編を参照してください。

5-2-1 基本データ型

データ型	内容	ビット	値の範囲
BOOL	ブール型	1	0 (FALSE) 、 1 (TRUE)
INT	整数型	16	-32768 ~ +32767
DINT	倍精度整数型	32	-2147483648 ~ +2147483647
LINT	4 倍精度整数型	64	-9223372036854775808 ~ +9223372036854775807
UINT	符号なし整数	16	&0 ~ 65535
UINT BCD	符号なし整数型 BCD		(注 1)
UDINT	符号なし倍精度整数型	32	&0 ~ 4294967295
UDINT BCD	符号なし倍精度整数型 BCD		(注 1)
ULINT	符号なし 4 倍精度整数型	64	&0 ~ 18446744073709551615
ULINT BCD	符号なし 4 倍精度整数型 BCD		(注 1)
REAL	実数型	32	-3.402823×10 ³⁸ ~-1.175494×10 ⁻³⁸ 、0、+1.175494×10 ⁻³⁸ ~+3.402823×10 ³⁸
LREAL (注 4)	倍精度実数型	64	-1.79769313486232×10 ³⁰⁸ ~-2.22507385850720×10 ⁻³⁰⁸ 、0、+2.22507385850720×10 ⁻³⁰⁸ ~+1.79769313486232×10 ³⁰⁸
WORD	長さ 16 のビット列型	16	#0000~FFFF または&0 ~ 65535
DWORD	長さ 32 のビット列型	32	#00000000~FFFFFFFF または&0 ~ 4294967295
LWORD	長さ 64 のビット列型	64	#0000000000000000~FFFFFFFFFFFFFFFF または&0 ~ 18446744073709551615
STRING (注 3)	文字列型	可変	—
FUCTION BLOCK	ファンクションブロック インスタンス	—	—
CHANNEL	チャンネル		(注 1)
NUMBER	定数、または番号		(注 2)
TIMER	タイマ型	アップフラグ:1 現在値:16	タイマ番号: 0~4095 タイムアップフラグ: 0、1 タイマ現在値: 0~9999 (BCD 方式)、 0~65535 (BIN 方式)
COUNTER	カウンタ型	アップフラグ:1 現在値:16	カウンタ番号: 0~4095 カウントアップフラグ: 0、1 カウンタ現在値: 0~9999 (BCD 方式)、 0~65535 (BIN 方式)

注 1: ST プログラムでは、以下のデータ型として認識されます。

・UINT BCD → WORD して認識されます。

5-2 ST プログラムで使用するデータ型

5-2-2 派生データ型

- ・ UDINT BCD → DWORD して認識されます。
- ・ ULINT BCD → LWORD して認識されます。
- ・ CHANNEL → WORD して認識されます。

注 2： ST プログラムでは使用できません。指定してもプログラムエラーとなります。

注 3： 入力方法については、「5-3 ST プログラムの入力について」を参照してください。

注 4： CP2E シリーズ PLC の ST プログラムでは使用できません。指定してもプログラムエラーとなります。

5-2-2 派生データ型

データ型	内容
配列データ型	1 次元配列。最大要素数: 32000
構造体データ型	ユーザ定義のデータ型

5-3 ST プログラムの入力について

5-3-1 記述上のルール

● 文の区切り

- ・文（代入文および制御文）は、必ず、半角セミコロン (;) で終わります。単に改行しただけでは、文の終わりとはなりませんのでご注意ください。
- ・文の予約語の後、値や式の後など、文の後以外に、半角セミコロン (;) は入れないようにしてください。入れた場合、文法エラーとなります。

● コメント

- ・コメントは、半角括弧と半角アスタリスクの、(* と*) で囲みます。コメント内には、(* と*) 以外の任意の文字を記述可能です。コメントの入れ子はできません。

表記

(* <コメント >*)

例

(* ここにコメントを記載します *)

注：コメントの入れ子は不可 (*(*これは不可*)*)

● スペース（空白）、改行、タブ入力について

- ・スペース（空白）、改行、タブ、またはそれらの任意の組み合わせを、任意の数、任意の位置に記述してもかまいません。
したがって、読みやすさを考慮して、予約語と式の間を改行したり、文の前にスペースやタブを入力することができます。
- ・ただし、以下の「トークン」（コンパイラに対して意味をもつ最小単位）と呼ぶ単位の途中に、スペース（空白）、改行、およびタブ（これらを「トークン分離符」と呼ぶ）を置くことはできません。

「トークン」：予約語、変数名、特殊記号、定数（数値）

予約語（大文字または小文字）：

AND、BY、CASE、DO、ELSE、ELSIF、EXIT、
FALSE、FOR、IF、NOT、OF、OR、REPEAT、
RETURN、THEN、TO、TRUE、UNTIL、WHILE、
XOR

変数名：

予約語以外の文字は、変数名とみなされます。

特殊記号（半角）：

<=、>=、<>、:=、..、&、(*、*)

定数（数値）

- ・10 進のときは、数値のみ
- ・16 進のときは、16#数値
- ・2 進のときは、2#数値
- ・8 進のときは、8#数値

もし、上記トークンの途中に、スペース（空白）、改行、タブを置くと、その前と後ろの文字の連続は、別のトークンとなります。

したがって、1つのトークンとしたいものの途中に、スペース（空白）、改行、タブを記述しないように注意してください。

- ・予約語と変数名の間には、スペース（空白）、改行、タブなどの「トークン分離符」を必ず入力してください。他のトークンとトークンの組み合わせの間には、「トークン分離符」を入力してもしなくてもかまいません。

例) ■は、スペース（空白）、改行、タブなどの「トークン分離符」を入力するべき場所です。

```
IF ■ A>0 THEN ■ X:=10;
ELSE ■
    X:=0;
END_IF;
```

● 小文字/大文字、半角/全角について

- ・予約語および変数名は、小文字、大文字を、区別しません（どちらでもかまいません）。
- ・予約語は、半角で入力してください。全角で入力すると、文法エラーとなります。

● 変数名の禁則文字について

- ・変数名には、次に [] 内に示す文字は使用できません。
[!], ["], [#], [\$], [%], [&], ['], [(], [)], [-], [=], [^], [~], [¥],
[] , [@], [], [], [{], [}], [;], [+], [:], [*], [], [}], [,], [<],
[.], [>], [/], [?]
 - ・変数名の先頭文字には、数字（0 ～9）は使用できません。
 - ・変数名に、複数のアンダーラインの連続は、使用できません
 - ・変数名に、スペース（空白）は使用できません
- これらの文字を使用すると、エラーメッセージが表示されます。

● 演算子の優先順について

- ・演算子は、優先順位を考慮して記述してください、または優先したい演算は、括弧（ ）でくるようにしてください。

例) 論理積の AND は、論理和の OR より優先される。そのため、「X OR Y AND Z」と記述すると、Y AND Z が優先される。

● STRING 型について

- ・サポートされる文字列は以下のとおりです。
半角英数字 最大 255 文字
大文字/小文字は区別される

・ ST 言語で定義された文字列は PLC のメモリには以下のように格納されます。

StrX := '123456'; の場合

n	31 32	文字列の最後には NULL(#00)が格納されます
n+1	33 34	
n+2	35 36	
n+3	00 00	

・ 文字列はシングルクォート(')で指定します。

表記	内容
'A'	文字列「A」(ASCII コード: #41)
' '	1 スペース文字(ASCII コード: #20)
' '	空文字(文字なし)

・ ドル記号(\$)の後ろに 2 桁の 16 進数を記載したものは、16 進表記と解釈します。

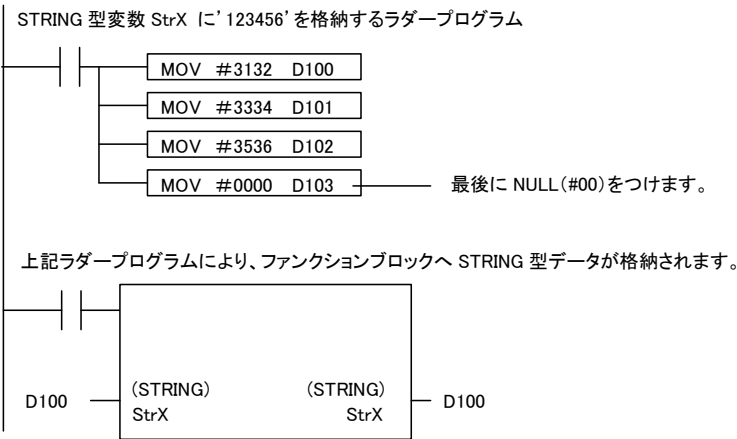
表記	内容
'\$02'	数値#02(スタートコード)
'\$03'	数値#03(エンドコード)

・ ドル記号(\$)の後ろにアルファベットを記載したものは、以下のように解釈します。

表記	内容
'\$\$'	ドル記号(ASCII コード: #24)
'\$'	シングルクォート(ASCII コード: #27)
'\$L'または'\$l'	改行(ASCII コード: #0A)
'\$N'または'\$n'	復帰改行(ASCII コード: #0D0A)
'\$P'または'\$p'	改ページまたは新しいページ(ASCII コード: #0C)
'\$R'または'\$r'	復帰(ASCII コード: #0D)
'\$T'または'\$t'	タブ(ASCII コード: #09)

・ ファンクションブロックなどの STRING 型変数に、ラダープログラムから文字列を格納する場合は、文字列の最後に NULL (#00) を付けてください。

ファンクションブロックの STRING 型変数 StrX に、文字列データを渡す場合



● TIMER 型/COUNTER 型について

TIMER 型/COUNTER 型の変数は以下のように記述します。

1) タイマ型変数の ST 内での記述方法

タイムアップフラグ : TIMER 型変数名.CF
タイム現在値 : TIMER 型変数名.PV
(例) タイムアップフラグ : Timer1.CF
タイム現在値 : Timer1.PV

2) カウンタ型変数の記述方法

カウントアップフラグ : COUNTER 型変数名.CF
カウンタ現在値 : COUNTER 型変数名.PV
(例) カウントアップフラグ : Counter1.CF
カウンタ現在値 : Counter1.PV

アップフラグは読出し可能ですが、書き込むことはできません。
現在値は、読出し、書き込みとも可能です。

5-3-2 CX-Programmer の ST 入力画面での表示

■ テキストの表示色

CX-Programmer では、ST 入力画面にテキストを入力または貼り付けると、以下の文字色で、自動的に表示されます。

- ・ 文のキーワード（予約語）：青色
- ・ コメント：緑色
- ・ エラー：赤色
- ・ 文字列：茶色
- ・ その他：黒色

■ フォントの変更

フォントサイズ、表示色を変更したい場合は、[ツール] メニューから [オプション] を選択して、[表示] タブで、[ST フォント] ボタンをクリックします。

フォント名、フォントサイズ（デフォルトは9ポイント）、色を、変更できます。

5-4 ST 言語の構成

5-4-1 文の一覧

文		機能	例
文の終わり		文を終了する。	;
コメント		(*と、*)で囲んだ部分がコメントとなる。	(* コメント *)
代入文	代入	左側の変数に、右側の式の結果、変数、値を代入する。	A:=B;
制御文	IF、THEN、ELSIF、ELSE、END_IF	条件式の評価結果が真のとき、文が実行される。	IF <条件式 1> THEN <文 1>; ELSIF <条件式 2> THEN <文 2>; ELSE <文 3>; END_IF;
	CASE、ELSE、END_CASE	整数式の値によって、文が選択される。	CASE <整数式> OF 1: <文 1>; 2: <文 2>; 3: <文 3>; ELSE <文 4>; END_CASE;
	FOR、TO、BY、DO、END_FOR	初期値、最終値、増加値の条件によって、文が繰り返し処理される。	FOR <変数> := <初期値> TO <最終値> BY <増加値> DO <文>; END_FOR;
	WHILE、DO、END_WHILE	条件式が真のあいだ、文が繰り返し処理される。	WHILE <条件式> DO <文>; END_WHILE;
	REPEAT、UNTIL、END_REPEAT	条件式が真になるまで、文が繰り返し処理される。	REPEAT <文>; UNTIL <条件式> END_REPEAT;
	EXIT	繰り返し処理を中止する。	EXIT;
	RETURN	<ST プログラム> 実行中の ST タスクを終了させ、次のタスクを実行します。 <SFC で使用する ST> 実行している SFC のアクションプログラムを終了させ、次のアクションプログラムを実行します。 <FB 内で使用する ST> 呼び出された自ファンクションブロックから、呼び出し元のプログラムの位置に戻ります。)	RETURN;
	ファンクションブロック 呼び出し文	ファンクションブロック定義を呼び出す。	<FB 内で使用する場合> データ型が FUNCTION BLOCK の変数名(呼び出し先ファンクションブロック定義の入力変数名:=自身の入力変数名または定数、...呼び出し先ファンクションブロック定義の出力変数名または定数=>自身の出力変数名、...);

5-4-2 変数について

変数についての仕様や設定方法については、CX-Programmer オペレーションマニュアル (No.SBCA-337) を参照してください。

5-4-3 定数入力について

10 進以外に、16 進、8 進、2 進で表記することができます。以下のように記載します。

	表記	例 (10 進で 12 の場合)
10 進数	数値を記載	12
16 進数	16#の後に数値を記載	16#C
8 進数	8#の後に数値を記載	8#14
2 進数	2#の後に数値を記載	2#1100
文字列	‘ ‘ (シングルコート) で囲む	‘Hello world’

お願い

16 進数、8 進数、2 進数の負数は整数の 2 の補数で表現します。

INT 型の有効範囲は 10 進数表記で「-32768～+32767」ですが、16 進数では「#0000～#FFFF」となるため、負数は整数の 2 の補数を使用します。

例) INT 型の変数に定数「-10」を 16 進数で設定する場合「16#FFF6」となります。

5

5-4-4 演算子一覧

演算	記号式	演算子で使えるデータ型	優先順位 1：最上位 11：最下位
括弧	(式), 配列[識別子]		1
関数	関数記号	関数による (付-3-6 参照)	2
べき乗 ^{*)}	**	底部：REAL、LREAL 指数部：INT、DINT、LINT、UINT、UDINT、ULINT、REAL、LREAL	3
補数	NOT	BOOL、WORD、DWORD、LWORD	4
乗算	*	INT、DINT、LINT、UINT、UDINT、ULINT、REAL、LREAL	5
除算	/	INT、DINT、LINT、UINT、UDINT、ULINT、REAL、LREAL	5
加算	+	INT、DINT、LINT、UINT、UDINT、ULINT、REAL、LREAL、STRING	6
減算	-	INT、DINT、LINT、UINT、UDINT、ULINT、REAL、LREAL	6
比較	<, >, <=, >=	BOOL、INT、DINT、LINT、UINT、UDINT、ULINT、WORD、DWORD、LWORD、REAL、LREAL、STRING	7
等式	=	BOOL、INT、DINT、LINT、UINT、UDINT、ULINT、WORD、DWORD、LWORD、REAL、LREAL、STRING	8
不等式	<>	BOOL、INT、DINT、LINT、UINT、UDINT、ULINT、WORD、DWORD、LWORD、REAL、LREAL、STRING	8
ブール AND	&	BOOL、WORD、DWORD、LWORD	9
ブール AND	AND	BOOL、WORD、DWORD、LWORD	9
ブール排他的論理和	XOR	BOOL、WORD、DWORD、LWORD	10
ブール OR	OR	BOOL、WORD、DWORD、LWORD	11

*1：CP2E シリーズ PLC では、使用できません。

・留意事項：演算は、データ型に応じて行われます。

したがって、たとえば、INT 型データの加算結果は、INT 型データの変数でなければなりません。とくに、整数型変数での演算の桁上がり、桁下がりご注意ください。
たとえば、INT 型の変数 A=3、B=2 で、(A/B)*2 を行くと、A/B の結果が 1 (1.5 の小数点以下切捨て) となり、(A/B)*2 は 2 となります。

5-4-5 標準関数一覧

関数種類	
数値演算関数	絶対値、三角関数などの数値演算関数
文字列関数	文字列抽出や結合などの標準文字列関数
データ型変更関数	変更元のデータ型_TO_変更先データ型 (変数名)
数値文字列変更関数	変更元のデータ型_TO_STRING (変数名) STRING_TO_変更先データ型 (変数名)
データシフト関数	ビットシフト、ビット回転などのデータシフト関数
データ制御関数	上下限リミット制御などの制御関数
データ選択関数	最大値/最小値、マルチプレクサなどのデータ選択関数

■ 数値演算関数

関数名	引数のデータ型		戻り値のデータ型	内容	例
ABS(引数) ^{*1}	INT DINT LINT UINT UDINT ULINT REAL LREAL		INT DINT LINT UINT UDINT ULINT REAL LREAL	絶対値 引数	a:=ABS(b) (* 変数 b の絶対値を変数 a に格納 *)
SQRT(引数) ^{*1}	REAL LREAL		REAL LREAL	平方根 $\sqrt{\text{引数}}$	a:= SQRT (b) (* 変数 b の平方根を変数 a に格納 *)
LN(引数) ^{*1}	REAL LREAL		REAL LREAL	自然対数 LOG _e 引数	a:= LN (b) (* 変数 b の自然対数を変数 a に格納 *)
LOG (引数) ^{*1}	REAL LREAL		REAL LREAL	常用対数 LOG ₁₀ 引数	a:= LOG (b) (* 変数 b の常用対数を変数 a に格納 *)
EXP(引数) ^{*1}	REAL LREAL		REAL LREAL	自然指数 e ^{引数}	a:= EXP (b) (* 変数 b の自然指数を変数 a に格納 *)
SIN(引数) ^{*1}	REAL LREAL		REAL LREAL	正弦 SIN 引数	a:= SIN (b) (* 変数 b の SIN を変数 a に格納 *)
COS(引数) ^{*1}	REAL LREAL		REAL LREAL	余弦 COS 引数	a:= COS (b) (* 変数 b の COS を変数 a に格納 *)
TAN(引数) ^{*1}	REAL LREAL		REAL LREAL	正接 TAN 引数	a:= TAN (b) (* 変数 b の TAN を変数 a に格納 *)
ASIN(引数) ^{*1}	REAL LREAL		REAL LREAL	逆正弦 SIN ⁻¹ 引数	a:= ASIN (b) (* 変数 b の ASIN を変数 a に格納 *)
ACOS(引数) ^{*1}	REAL LREAL		REAL LREAL	逆余弦 COS ⁻¹ 引数	a:= ACOS(b) (* 変数 b の ACOS を変数 a に格納 *)
ATAN(引数) ^{*1}	REAL LREAL		REAL LREAL	逆正接関数 TAN ⁻¹ 引数	a:= ATAN(b) (* 変数 b の ATAN を変数 a に格納 *)
EXPT(<底部>, <指数部>) ^{*1}	底部	REAL LREAL	REAL LREAL	指数 底部 ^{指数部}	a:= EXPT (b,c) (* 変数 b を底部、変数 c を指数部とした指数を変数 a に格納 *)
	指数部	INT DINT LINT UINT UDINT ULINT REAL LREAL			
MOD(<除算対象データ>,<除算データ>)	除算対象データ	INT UINT UDINT ULINT DINT LINT	INT UINT UDINT ULINT DINT LINT	剰余	a:= MOD(b, c) (* 変数 b を変数 c で割った剰余を変数 a に格納*)
	除算データ	INT UINT UDINT ULINT DINT LINT			

注：数値演算関数では、引数と同じデータ型が返ります。したがって、関数の戻り値を代入する変数は、引数と同じデータ型とすることが必要です。

*1：CP2E シリーズ PLC では、使用できません。

■ 文字列関数

以下の関数は CS/CJ シリーズユニット Ver.4.0 以降、および CJ2 シリーズの CPU ユニットで使用する
ことができます。

関数名	引数のデータ型		戻り値の データ型	内容	例
LEN(文字列)	文字列	STRING	INT	文字列長の検出	a:= LEN (b) (* 変数 b の文字列長を変数 a に格納 *)
LEFT(<抽出対 象文字列>, <抽 出文字数>)	抽出対象 文字列	STRING	STRING	左からの文字列抽出	a:= LEFT(b,c) (* 変数 b の文字列の左端から変数 c で指定 した文字数分だけ変数 a に格納 *)
	抽出文字 数	INT UINT			
RIGHT(<抽出対 象文字列>, <抽 出文字数>)	抽出対象 文字列	STRING	STRING	右からの文字列抽出	a:= RIGHT(b,c) (* 変数 b の文字列の右端から変数 c で指定 した文字数分だけ変数 a に格納 *)
	抽出文字 数	INT UINT			
MID(<抽出対象 文字列>, <抽出 文字数>, <抽出 位置>)	抽出対象 文字列	STRING	STRING	文字列の抽出	a:= MID(b,c,d) (* 変数 b の文字列の変数 d で指定した位置 から変数 c で指定した文字数分だけ変数 a に格納 *)
	抽出文字 数	INT UINT			
	抽出位置	INT UINT			
CONCAT(<結合 文字列 1>, <結 合文字列 2>, ...) *最大 31 個まで 指定可能	結合文字 列	STRING	STRING	文字列の結合	a:= CONCAT(b,c,...) (* 変数 b、変数 c の文字列を結合して変数 a に格納 *)
INSERT(<挿入 対象文字列>, < 挿入文字列>, < 挿入位置>)	挿入対象 文字列	STRING	STRING	文字列の挿入	a:= INSERT(b,c,d) (* 変数 b の変数 d で指定した位置に変数 c の文字列を挿入して変数 a に格納 *)
	挿入文字 列	STRING			
	挿入位置	INT UINT			
DELETE(<削除 対象文字列>, < 削除文字数>, < 削除位置>)	削除対象 文字列	STRING	STRING	文字列の削除	a:= DELETE(b,c,d) (* 変数 b の変数 d で指定した位置から変数 c で指定した文字数分だけ削除して変数 a に格納 *)
	削除文字 数	INT UINT			
	削除位置	INT UINT			
REPLACE(<置 換対象文字列>, <置換文字列>, < 置換文字数>, < 置換位置>)	置換対象 文字列	STRING	STRING	文字列の置換	a:= REPLACE(b,c,d,e) (* 変数 b の変数 e で指定した位置から変数 d で指定した文字数分だけ変数 c で置換し て変数 a に格納 *)
	置換文字 列	STRING			
	置換文字 数	INT UINT			
	置換位置	INT UINT			
FIND(<検索対象 文字列>, <検索 文字列>)	検索対象 文字列	STRING	INT	文字列の検索	a:= FIND(b,c) (* 変数 b の文字列から変数 c で指定された 文字列を検索し、最初に見つかった文字位 置を変数 a に格納。見つからなかった場合 は変数 a に 0 を格納。 *)
	検索文字 列	STRING			

■ データ型変更関数

ST 言語内で、以下のデータ型変更関数を使用可能です。

● 構文：

変更元のデータ型_TO_変更先データ型（変数名）

例) REAL_TO_INT (C)

変数 C のデータ型を、REAL 型から INT 型へ変更します。

変数 C の値の小数点以下は、五捨五入して整数に丸めます。五捨五入とは下の表のような処理のことです。

小数部の値	処理	例	
0.5 より小さい	切り捨て	1.49 → 1	-1.49 → -1
0.5	1 の位の値が偶数：切り捨て 1 の位の値が奇数：切り上げ	0.5 → 0 1.5 → 2 2.5 → 2 3.5 → 4	-0.5 → 0 -1.5 → -2 -2.5 → -2 -3.5 → -4
0.5 より大きい	切り上げ	1.51 → 2	-1.51 → -2

● データ型の組み合わせ

データ型の変更元 (FROM) と変更先 (TO) の組み合わせの可否は以下のとおりです。

○：可能、×：不可

TO \ FROM	BOOL	INT	DINT	LINT	UINT	UDINT	ULINT	WORD	DWORD	LWORD	REAL	LREAL	BCD_WORD	BCD_DWORD
BOOL	×	×	×	×	×	×	×	×	×	×	×	×	×	×
INT	×	×	○	○	○	○	○	○	○	○	○	○	○	○
DINT	×	○	×	○	○	○	○	○	○	○	○	○	○	○
LINT	×	○	○	×	○	○	○	○	○	○	○	○	×	×
UINT	×	○	○	○	×	○	○	○	○	○	○	○	○	○
UDINT	×	○	○	○	○	×	○	○	○	○	○	○	○	○
ULINT	×	○	○	○	○	○	×	○	○	○	○	○	×	×
WORD	×	○	○	○	○	○	○	×	○	○	×	×	×	×
DWORD	×	○	○	○	○	○	○	○	×	○	×	×	×	×
LWORD	×	○	○	○	○	○	○	○	○	×	×	×	×	×
REAL	×	○	○	○	○	○	○	×	×	×	×	○	×	×
LREAL	×	○	○	○	○	○	○	×	×	×	○	×	×	×
WORD_BCD	×	○	○	×	○	○	×	×	×	×	×	×	×	×
DWORD_BCD	×	○	○	×	○	○	×	×	×	×	×	×	×	×

■ 数値文字列変更関数

ST 言語内で、以下の数値文字列変更関数を使用可能です。

● 構文：

変更元のデータ型_TO_STRING（変数名）

例) INT_TO_STRING (C)

変数 C のデータ型を、INT 型から STRING 型へ変更します。

STRING_TO_変更先のデータ型（変数名）

例) STRING_TO_INT (C)

変数 C のデータ型を、STRING 型から INT 型へ変更します。

5-4 ST 言語の構成

5-4-5 標準関数一覧

● データ型の組み合わせ

データ型の変更元 (FROM) と変更先 (TO) の組み合わせの可否は以下のとおりです。

○：可能、×：不可

TO \ FROM	BOOL	INT	DINT	LINT	UINT	UDINT	ULINT	WORD	DWORD	LWORD	REAL	LREAL	STRING
BOOL	×	×	×	×	×	×	×	×	×	×	×	×	×
INT	×	×	○	○	○	○	○	○	○	○	○	○	○
DINT	×	○	×	○	○	○	○	○	○	○	○	○	○
LINT	×	○	○	×	○	○	○	○	○	○	○	○	×
UINT	×	○	○	○	×	○	○	○	○	○	○	○	○
UDINT	×	○	○	○	○	×	○	○	○	○	○	○	○
ULINT	×	○	○	○	○	○	×	○	○	○	○	○	×
WORD	×	○	○	○	○	○	○	×	○	○	×	×	○
DWORD	×	○	○	○	○	○	○	○	×	○	×	×	○
LWORD	×	○	○	○	○	○	○	○	○	×	×	×	×
REAL	×	○	○	○	○	○	○	○	○	○	×	×	×
LREAL	×	○	○	○	○	○	○	○	○	○	×	×	×
STRING	×	○	○	×	○	○	×	○	○	×	×	×	×

■ データシフト関数

関数名	第 1 引数のデータ型	第 2 引数のデータ型	戻り値のデータ型	内容	例
SHL(<シフト対象データ>, <ビット数>)	BOOL, WORD, DWORD, LWORD	INT, UINT, UDINT, ULINT, DINT, LINT,	BOOL, WORD, DWORD, LWORD	ビット文字列を左へ n ビットシフトします。シフトしたとき、右にはゼロが入ります。	a := SHL(b, c) (*ビット文字列 b を左へ c ビットシフトした結果を a に格納*)
SHR(<シフト対象データ>, <ビット数>)	BOOL, WORD, DWORD, LWORD	INT, UINT, UDINT, ULINT, DINT, LINT,	BOOL, WORD, DWORD, LWORD	ビット文字列を右へ n ビットシフトします。シフトしたとき、左にはゼロが入ります。	a := SHR(b, c) (*ビット文字列 b を右へ c ビットシフトした結果を a に格納*)
ROL(<回転対象データ>, <ビット数>)	BOOL, WORD, DWORD, LWORD	INT, UINT, UDINT, ULINT, DINT, LINT,	BOOL, WORD, DWORD, LWORD	ビット文字列を左へ n ビット回転します。	a := ROL(b, c) (*ビット文字列 b を左へ c ビット回転した結果を a に格納*)
ROR(<回転対象データ>, <ビット数>)	BOOL, WORD, DWORD, LWORD	INT, UINT, UDINT, ULINT, DINT, LINT,	BOOL, WORD, DWORD, LWORD	ビット文字列を右へ n ビット回転します。	a := ROR(b, c) (*ビット文字列 b を右へ c ビット回転した結果を a に格納*)

■ データ制御関数

関数名	第 1 引数のデータ型	第 2 引数のデータ型	第 3 引数のデータ型	戻り値のデータ型	内容	例
LIMIT(<下限りミットデータ>, <入力データ>, <上限りミットデータ>)	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	入力データが上下限りミットデータの範囲内かどうかにより、出力データを制御します。	a := LIMIT(b, c, d) (*c < b のとき b を、b ≤ c ≤ d のとき c を、d < c のとき d を、a に格納*)

■ データ選択関数

関数名	第1引数のデータ型	第2引数のデータ型	第3引数のデータ型*1	戻り値のデータ型	内容	例
SEL(<選択条件>, <選択対象データ 1>, <選択対象データ 2>)	BOOL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	2 つのデータから選択条件で指定された 1 つのデータを選択します。	a := SEL(b, c, d) (*c と d のうち b が偽ならば c を、真ならば d を a に格納する*)
MUX(<抽出条件>, <抽出対象データ 1>, <抽出対象データ 2>, …)	INT, UINT, UDINT, ULINT, DINT, LINT	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	最大 30 個のデータから抽出条件で指定された 1 つのデータを選択します。	a := MUX(b, c, d, …) (*c、d、…の中から b+1 番目のデータを a に格納します*)
MAX(<対象データ 1>, <対象データ 2>, <対象データ 3>, …) *2	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	最大 31 個のデータから、最大値を選択します。	a := MAX(b, c, d, …) (*c, d, …の中から最大値を a に格納します。*)
MIN(<対象データ 1>, <対象データ 2>, <対象データ 3>, …) *2	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	最大 31 個のデータから、最小値を選択します。	a := MIN(b, c, d, …) (*c, d, …の中から最小値を a に格納します。*)

*1：MUX は第 3 引数以降、最大で第 31 引数（最大 30 個の抽出対象データ）まで指定可能です。

*2：MAX、MIN は、第 1 引数から第 31 引数（最大 31 個の対象データ）の間で引数を指定可能です。

5-4-6 オムロン拡張関数一覧

関数種類	
メモ리카ード処理関数	メモ리카ードへ書き込む関数
通信処理関数	文字列を送受信する関数
角度変換関数	角度とラジアンを変換する関数
タイマ/カウンタ関数	様々なタイマ/カウンタを実行する関数

■ メモ리카ード処理関数

以下の関数は CS/CJ シリーズユニット Ver.4.0 以降、および CJ2 シリーズの CPU ユニットで 사용할 수 있습니다。

関数名	引数のデータ型		戻り値のデータ型	内容	例
WRITE_TEXT(<書出文字列>,<ディレクトリ名+ファイル名>,<区切り文字>,<パラメータ>)	書出文字列	STRING	-	メモ리카ードへの文字列書き込み	WRITE_TEXT(a,b,c,d) (* 変数 a で指定した文字列を変数 b で指定したディレクトリにテキストファイルとして保存する。変数 d が 0 の時は追加書き込み、1 の時はファイルを新規作成して書き込む *)
	ディレクトリ名+ファイル名	STRING			
	区切り文字	STRING			
	パラメータ	INT UINT WORD			

■ 通信処理関数

以下の関数は CS/CJ シリーズユニット Ver.4.0 以降、および CJ2 シリーズの CPU ユニットで 사용할 수 있습니다。

関数名	引数のデータ型		戻り値のデータ型	内容	例
TXD_CPU(<送信文字列>)	送信文字列	STRING	-	文字列の送信 (CPU ユニットの RS-232C ポート)	TXD_CPU(a) (* 変数 a で指定した文字列を CPU ユニットの RS-232C ポートから送信する *)
TXD_SCB(<送信文字列>,<シリアルポート No.>)	送信文字列	STRING	-	文字列の送信 (SCB のシリアルポート)	TXD_SCB(a,b) (* 変数 a で指定した文字列を変数 b で指定した SCB のシリアルポートから送信する *)
	シリアルポート No	INT UINT WORD			
TXD_SCU(<送信文字列>,<指定 SCU ユニット番号>,<シリアルポート No.>,<内部論理ポート No.>)	送信文字列	STRING	-	文字列の送信 (SCU のシリアルポート)	TXD_SCU(a,b,c,d) (* 変数 a で指定した文字列を変数 b で指定したユニット番号の SCU の変数 c で指定したシリアルポートから送信する。変数 d は内部論理ポート番号です *)
	指定 SCU ユニット番号	INT UINT WORD			
	シリアルポート No	INT UINT WORD			
	内部論理ポート No	INT UINT WORD			
RXD_CPU(<受信文字列格納先>,<受信文字数>)	受信文字列格納先	STRING	-	文字列の受信 (CPU ユニットの RS-232C ポート)	RXD_CPU(a,b) (* CPU ユニットの RS-232C ポートに受信したデータを変数 b の文字数分の文字列として変数 a に出力する *)
	受信文字数	INT UINT WORD			
RXD_SCB(<受信文字列格納先>,<受信文字数>,<シリアルポート No.>)	受信文字列格納先	STRING	-	文字列の受信 (SCB のシリアルポート)	RXD_SCB(a,b,c) (* 変数 b で指定した SCB のシリアルポートに受信したデータを変数 c の文字数分の文字列として変数 a に出力する *)
	受信文字数	INT UINT WORD			
	シリアルポート No	INT UINT WORD			

関数名	引数のデータ型		戻り値のデータ型	内容	例
RXD_SCU(<受信文字列格納先>,<受信文字数>,<指定 SCU ユニット番号>,<シリアルポート No.>,<内部論理ポート No.>)	受信文字列格納先	STRING	-	文字列の受信 (SCU のシリアルポート)	RXD_SCU(a,b,c,d,e) (* 変数 c で指定したユニット番号の SCU の変数 d で指定したシリアルポートに受信したデータを変数 b の文字数分の文字列として変数 a に出力する。変数 e は内部論理ポート番号です *)
	受信文字数	INT UINT WORD			
	指定 SCU ユニット番号	INT UINT WORD			
	シリアルポート No	INT UINT WORD			
	内部論理ポート No	INT UINT WORD			

■ 角度変換関数

以下の関数は CS/CJ シリーズユニット Ver.4.0 以降、および CJ2 シリーズの CPU ユニットで使用することができます。

関数名	引数のデータ型	戻り値のデータ型	内容	例
DEG_TO_RAD (引数)	REAL LREAL	REAL LREAL	角度→ラジアン変換	a:= DEG_TO_RAD(b) (* 変数 b (角度) をラジアンに変換して変数 a に格納 *)
RAD_TO_DEG (引数)	REAL LREAL	REAL LREAL	ラジアン→角度変換	a:= RAD_TO_DEG(b) (* 変数 b (ラジアン) を角度に変換して変数 a に格納 *)

■ タイマ/カウンタ関数

以下の関数は、CJ2 シリーズの CPU ユニットで使用することができます。

関数名	引数のデータ型		戻り値のデータ型	内容	例
TIMX(<実行条件>,<タイマアドレス>,<タイマ設定値>)	実行条件	BOOL	-	名称: 100ms タイマ 動作: 減算式 ON デイレー100ms 単位のタイマの動作をします。	TIMX(a, b, c) (*実行条件 a が成立した時点で、タイマアドレス b、設定値 c の TIMX タイマの実行を開始します。*)
	タイマアドレス	TIMER			
	タイマ設定値	UINT			
TIMHX(<実行条件>,<タイマアドレス>,<タイマ設定値>)	実行条件	BOOL	-	名称: 10ms タイマ (高速タイマ) 動作: 減算式 ON デイレー10ms 単位のタイマの動作をします。	TIMHX(a, b, c) (*実行条件 a が成立した時点で、タイマアドレス b、設定値 c の TIMHX タイマの実行を開始します。*)
	タイマアドレス	TIMER			
	タイマ設定値	UINT			
TMHHX(<実行条件>,<タイマアドレス>,<タイマ設定値>)	実行条件	BOOL	-	名称: 1ms タイマ (超高速タイマ) 動作: 減算式 ON デイレー1ms 単位のタイマの動作をします。	TMHHX(a, b, c) (*実行条件 a が成立した時点で、タイマアドレス b、設定値 c の TMHHX タイマの実行を開始します。*)
	タイマアドレス	TIMER			
	タイマ設定値	UINT			
TIMUX(<実行条件>,<タイマアドレス>,<タイマ設定値>)	実行条件	BOOL	-	名称: 0.1ms タイマ 動作: 減算式 ON デイレー0.1ms 単位のタイマの動作をします。	TIMUX(a, b, c) (*実行条件 a が成立した時点で、タイマアドレス b、設定値 c の TIMUX タイマの実行を開始します。*)
	タイマアドレス	TIMER			
	タイマ設定値	UINT			
TMUHX(<実行条件>,<タイマアドレス>,<タイマ設定値>)	実行条件	BOOL	-	名称: 0.01ms タイマ 動作: 減算式 ON デイレー0.01ms 単位のタイマの動作をします。	TMUHX(a, b, c) (*実行条件 a が成立した時点で、タイマアドレス b、設定値 c の TMUHX タイマの実行を開始します。*)
	タイマアドレス	TIMER			
	タイマ設定値	UINT			

5-4 ST 言語の構成
5-4-6 オムロン拡張関数一覧

関数名	引数のデータ型		戻り値のデータ型	内容	例
TTIMX(<実行条件>, <リセット入力>, <タイマアドレス>, <タイマ設定値>)	実行条件	BOOL	—	名称：積算タイマ 動作：積算式 ON デイレー 0.1s 単位のタイマの動作をします。	TTIMX(a, b, c, d) (*実行条件 a が ON の間、タイマアドレス c、設定値 d の TTIMX タイマを実行します。リセット入力 b が ON の場合に、タイマ現在値、アップフラグがリセットされます。*)
	リセット入力	BOOL			
	タイマアドレス	TIMER			
	タイマ設定値	UINT			
CNTX(<カウンタ入力>, <リセット入力>, <カウンタアドレス>, <カウンタ設定値>)	カウンタ入力	BOOL	—	名称：カウンタ 動作：減算カウンタの動作をします。	CNTX(a, b, c, d) (*カウンタ入力 a の立ち上がりごとに、カウンタアドレス c、カウンタ設定値 d の CNTX カウンタを実行します。リセット入力 b が ON のとき、カウンタ現在値、アップフラグがリセットされます。*)
	リセット入力	BOOL			
	カウンタアドレス	COUNTER			
	カウンタ設定値	UINT			
CNTRX(<加算カウンタ>, <減算カウンタ>, <リセット入力>, <カウンタアドレス>, <カウンタ設定値>)	加算カウンタ	BOOL	—	名称：可逆カウンタ 動作：加減算カウンタの動作をします。	CNTRX(a, b, c, d, e) (*カウンタアドレス d、カウンタ設定値 e の CNTRX を実行します。加算カウンタ入力 a の立ち上がりで現在値を加算し、減算カウンタ入力 b の立ち上がりで減算します。リセット入力 c が ON のとき、カウンタ現在値、アップフラグがリセットされます。*)
	減算カウンタ	BOOL			
	リセット入力	BOOL			
	カウンタアドレス	COUNTER			
	カウンタ設定値	UINT			
TRSET(<実行条件>, <タイマアドレス>)	実行条件	BOOL	—	名称：タイマリセット 動作：指定タイマをリセットします。	TRSET(a, b) (*実行条件 a が ON のときにタイマアドレス b のタイマをリセットします。*)
	タイマアドレス	TIMER			

5-5 各構文の説明

5-5-1 代入文

- ・ 概要：

左辺（変数）に、右辺（式、変数、または定数）を代入します。

- ・ 予約語：

:=

コロン（:）とイコール（=）の組み合わせ

- ・ 文の構造：

<変数>:=<式、変数、または定数>;

- ・ 用途：

変数に値を入力するときに、使用します。

制御文の前、または制御文の中の<文>に記述する、基本的な文です。

例えば、初期値の設定、計算結果の格納、変数のインCREMENT・デCREMENTなどに使用します。

- ・ 説明：

<式、変数、または定数>を、<変数>に、代入（格納）します。

- ・ 例：

例 1) 式 $X+1$ の結果を、変数 A に代入する

A:=X+1;

例 2) 変数 B の値を、変数 A に代入する

A:=B;

例 3) 定数 10 を、変数 A に代入する

A:=10;

- ・ 留意事項：

代入する式、変数、または定数のデータ型は、代入する変数のデータ型と一致している必要があります。もし一致していない場合、文法エラーとなります。

5-5-2 制御文

● IF 文（1 つの条件）

- 概要：

ある条件が成立したとき、ある文を実行します。成立しないとき、別の文を実行します。

- 予約語：

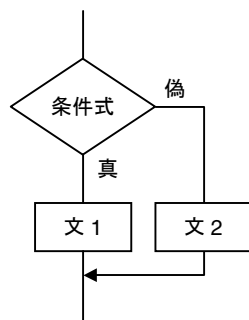
IF、THEN、（ELSE）、END_IF

注：ELSE は省略可能。

- 文の構造：

```
IF <条件式> THEN
    <文 1>;
ELSE
    <文 2>;
END_IF;
```

- 処理フロー図：



- 用途：

ある 1 つの条件（条件式）が成立したか否かで場合分けするときに、使用します。

- 説明：

<条件式> = 真のとき、<文 1>を実行します。

<条件式> = 偽のとき、<文 2>を実行します。

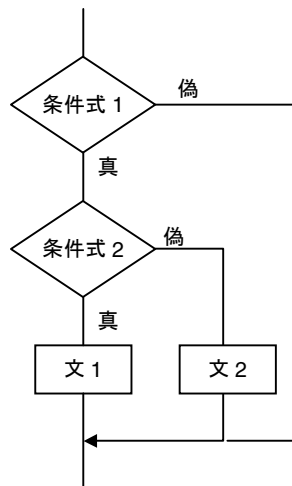
・ 留意事項：

- ・ IF は、必ず、END_IF と対で使用することが必要です。
- ・ <条件式> には、評価結果が真または偽の式を記述します。例) IF(A>10)
 なお、<条件式> には、式でなく、BOOL 型の変数のみを指定することもできます。
 その場合、変数の値が 1 (ON) = 評価結果真、0 (OFF) = 評価結果偽です。
- ・ <文 1>、<文 2>に、代入文、IF 文、CASE 文、FOR 文、WHILE 文、または REPEAT 文が可能です。

例) <文 1>に、さらに IF 文を入れた場合

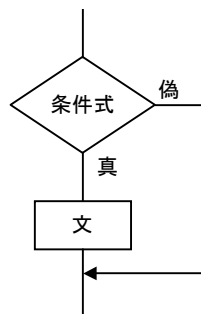
```
IF <条件式 1> THEN
  IF <条件式 2> THEN
    <文 1>;
  ELSE
    <文 2>;
  END_IF;
END_IF;
```

処理フロー図：



なお、ELSE は、上記のように、直前の THEN に対応します。

- ・ <文 1>、<文 2>の中は、ともに複数の文を実行可能です。その場合、文と文の間を;
 で区切ります。
- ・ ELSE 文は省略可能です。省略した場合、<条件式>=偽のとき、何も実行しません。
- ・ 処理フロー図：



・ 例：

例 1) 変数 A>0 が真のとき、変数 X に数値 10 を代入する。変数 A>0 が偽のとき、変数 X に数値 0 を代入する。

```
IF A>0 THEN
  X:=10;
ELSE
  X:=0;
END_IF;
```

例 2) 変数 A>0 かつ変数 B>1 が真のとき、変数 X に数値 10 を代入し、変数 Y に数値 20 を代入する。変数 A>0 かつ変数 B>1 が偽のとき、変数 X に数値 0 を代入し、変数 Y に数値 0 を代入する。

```
IF A>0 AND B>1 THEN
  X:=10;Y:=20;
ELSE
  X:=0;Y:=0;
END_IF;
```

例 3) BOOL 型の変数 A = 1 (ON) のとき、変数 X に数値 10 を代入する。変数 A = 0 (OFF) が偽のとき、変数 X に数値 0 を代入する。

```
IF A THEN X:=10;
ELSE X:=0;
END_IF;
```

● IF 文（複数の条件）

・ 概要：

ある条件が成立したとき、ある文を実行します。成立しないとき、さらに別の条件が成立したら、別の文を実行します。いずれも成立しないとき、別の文を実行します。

・ 予約語：

IF、THEN、ELSIF、(ELSE)、END_IF

注：ELSE は省略可能。

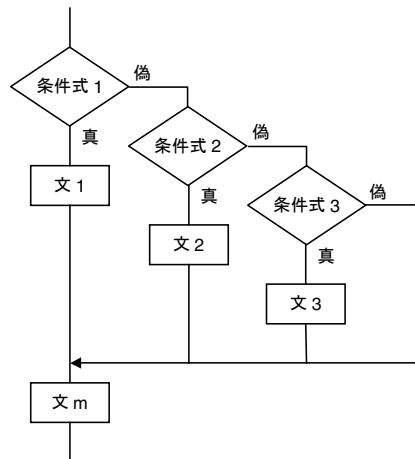
・ 文の構造：

```
IF <条件式 1> THEN <文 1>;
ELSIF <条件式 2> THEN <文 2>;
ELSIF <条件式 3> THEN <文 3>;

...

ELSIF <条件式 n> THEN <文 n>;
ELSE<文 m>;
END_IF;
```

・ 処理フロー図：



・ 用途：

ある複数の条件（条件式）が成立したか否かで場合分けするときに、使用します。

・ 説明：

<条件式 1>=真のとき、<文 1>を実行します。

<条件式 1>=偽のとき、

<条件式 2>=真なら、<文 2>を実行します。

<条件式 2>=偽のとき、

<条件式 3>=真なら、<文 3>を実行します。

⋮

<条件式 n>=真なら、<文 n>を実行します。

いずれの条件にも合致しないとき、<文 m>を実行します。

・ 留意事項：

・ IF は、必ず、END_IF と対で使用することが必要です。

・ <条件式□> には、評価結果が真または偽の式を記述します。例) IF(A>10)

なお、<条件式□> には、式でなく、BOOL 型の変数のみを指定することもできます。

その場合、変数の値が 1 (ON) = 評価結果真、0 (OFF) = 評価結果偽です。

・ <文□>に、代入文、IF 文、CASE 文、FOR 文、WHILE 文、または REPEAT 文が可能です。

・ <文□>の中で、複数の文を実行可能です。その場合、文と文の間を;で区切ります。

・ ELSE 文は省略可能です。省略した場合、いずれの条件にも合致しないとき、何も実行しません。

・ 例：

変数 A>0 が真のとき、変数 X に数値 10 を代入する。

変数 A>0 が偽のとき、変数 B=1 が真なら、変数 X に数値 1 を代入する。

変数 A>0 が偽のとき、変数 B=2 が真なら、変数 X に数値 2 を代入する。

いずれの条件にも合致しないとき、変数 X に数値 0 を代入する。

```

IF A>0 THEN X:=10;
    ELSIF B=1 THEN X:=1;
    ELSIF B=2 THEN X:=2;
ELSE X:=0;
END_IF;
  
```

● CASE 文

・ 概要：

整数式の値に一致する整数の選択値をもった文を実行します。

一致する整数の選択値がないとき、何も実行しない、またはある文を実行します。

・ 予約：

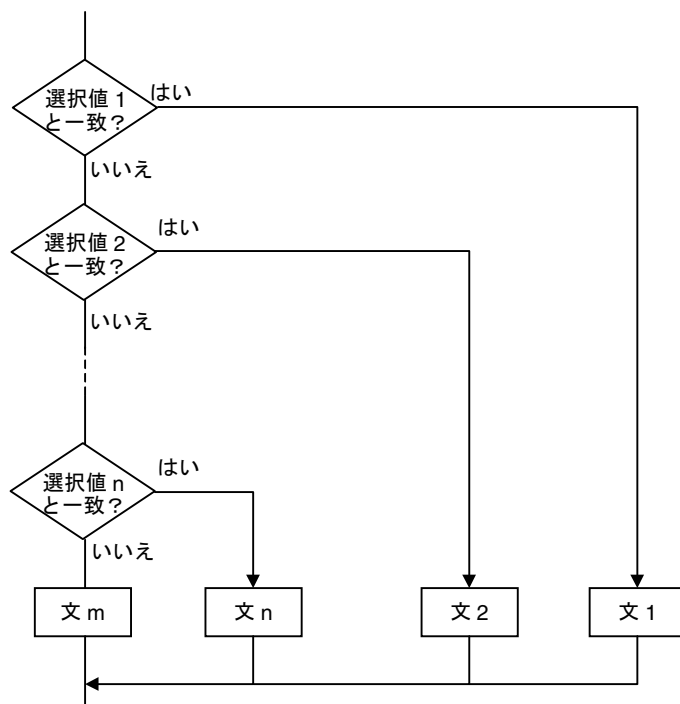
CASE

・ 文の構造：

```

CASE <整数式> OF
    <整数式の値 1>:<文 1>;
    <整数式の値 2>:<文 2>;
    ...
    <整数式の値 n>:<文 n>;
ELSE<文 m>;
END_CASE;
  
```

・ 処理フロー図：



・ 用途：

ある整数の値によって場合分けするときに、使用します。

・ 説明：

<整数式>が<整数式の値 n>と一致したとき、<文 n>を実行します。

<整数式>がいずれの<整数式の値 n>と一致しないとき、<文 m>を実行します。

・ 留意事項：

- ・ CASE は、必ず、END_CASE と対で使用することが必要です。
- ・ <整数式>の演算結果は、整数型（INT、DINT、LINT、UINT、UDINT、または ULINT）であることが必要です。
- ・ <文□>に、代入文、IF 文、CASE 文、FOR 文、WHILE 文、または REPEAT 文が可能です。
- ・ <文□>の中で、複数の文を実行可能です。その場合、文と文の間を;で区切ります。
- ・ <整数式> には、整数型（INT、DINT、LINT、UINT、UDINT、または ULINT）の変数、または整数値を返す式を、指定することが可能です。
- ・ <整数式の値 n> で、複数の整数の OR 論理を指定するときは、数値を、(コンマ)で区切ります。連続する整数を指定するときは、開始整数と終了整数の間を..（ピリオドを 2 つ）で区切ります。

・ 例：

例 1) 変数 A が 1 のとき、変数 X に数値 1 を代入する。変数 A が 2 のとき、変数 X に数値 2 を代入する。変数 A が 3 のとき、変数 X に数値 3 を代入する。いずれにも一致しないときは、変数 Y に 0 を代入する。

```
CASE A OF
```

```
1: X:=1;
```

```
2: X:=2;
```

```
3: X:=3;
```

```
ELSE Y:=0;
```

```
END_CASE;
```

例 2) 変数 A が 1 のとき、変数 X に数値 1 を代入する。変数 A が 2 または 5 のとき、変数 X に数値 2 を代入する。変数 A が 6～10 のいずれかのとき、変数 X に数値 3 を代入する。変数 A が 11、12 または 15～20 のとき、変数 X に数値 4 を代入する。いずれにも一致しないときは、変数 Y に 0 を代入する。

```
CASE A OF
```

```
1: X:=1;
```

```
2,5: X:=2;
```

```
6..10: X:=3;
```

```
11,12,15..20: X:=4;
```

```
ELSE Y:=0;
```

```
END_CASE;
```

● FOR 文

・ 概要：

変数（以下これを「反復変数」と呼ぶ）がある値からある値になるまで、ある文を繰り返して実行します。

・ 予約語：

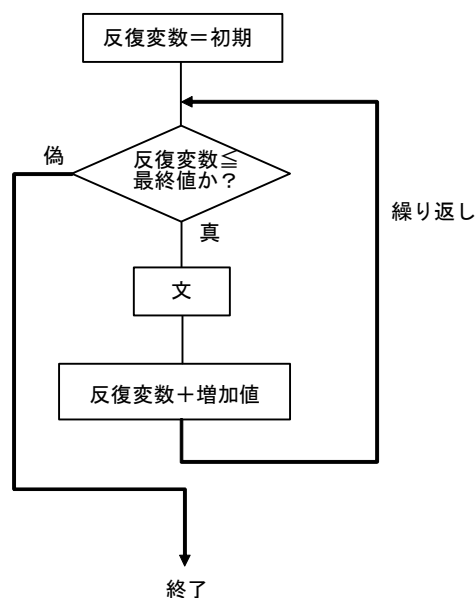
FOR、TO、（BY）、DO、END_FOR

注：BY は省略可能。

・ 文の構造：

```
FOR <反復変数> := <初期値> TO <最終値の式> BY <増加値の式>
DO
  <文>;
END_FOR;
```

・ 処理フロー図：



・ 用途：

繰り返し回数があらかじめわかっている場合に使用します。

とくに、ある反復変数の値に応じて、配列型変数の各要素を切り替えて指定する場合に、有効です。

・ 説明：

<反復変数>が<初期値>の値のとき、<文>を実行します。実行後、<反復変数>に<増加値の式>で与えられる値を加算し、<反復変数>≤<最終値の式>の値であれば*1、<文>を実行します。実行後、<反復変数>に<増加値の式>で与えられる値を加算し、<反復変数>≤<最終値の式>の値であれば*1、<文>を実行します。これを繰り返します。

<反復変数>><最終値の式>となると*2、終了します。

*1：<増加値の式>がマイナスの値の場合は、「<反復変数>≥<最終値の式>の値であれば」。

*2：<増加値の式>がマイナスの値の場合は、「<反復変数><<最終値の式>となると」。

・ 留意事項：

- ・ <増加値の式>に、マイナスの値を指定することができます。
- ・ FOR は、必ず、END_FOR と対で使用することが必要です。
- ・ <初期値>、<最終値の式>、<増加値の式>は、整数型（INT、DINT、LINT、UINT、UDINT、または ULINT）であることが必要です。
- ・ 反復変数では最終値で処理を実行した後、（最終値+1）となり、反復処理を終了します。

例）以下のような ST 文では a の値が TRUE となります。

```
FOR i:=0 TO 100 DO
  DATA[i]:=0;
END_FOR;

IF i=101 THEN
  a:=TRUE;
ELSE
  a:=FALSE;
END_IF;
```

- ・ FOR 文の内側で反復変数を直接変更するような処理は記述しないでください。予期せぬ動作となる場合があります。

例）

```
FOR i:=0 TO 100 BY 1 DO
  DATA[i]:=0;
  i:=i+5;
END_FOR;
```

- ・ <文>に、代入文、IF 文、CASE 文、FOR 文、WHILE 文、または REPEAT 文が可能です。
- ・ <文>の中で、複数の文を実行可能です。その場合、文と文の間を;で区切ります。
- ・ BY<増加値の式>は省略可能です。省略した場合、BY は 1 として実行されます。
- ・ <初期値>、<最終値の式>、<増加値の式>には、整数型（INT、DINT、LINT、UINT、UDINT、または ULINT）の変数、または整数値を返す式を、指定することが可能です。

例 1) 反復変数 n=0 から 5 つおきに 50 まで、繰り返し、配列変数 SP[n]に 100 を代入します。

```
FOR n := 0          TO 50   BY 5 DO
  SP[n] := 100;
END_FOR;
```

例 2) 配列変数 DATA[n] の要素の DATA[1] から DATA[50] の合計値を算出し、変数 SUM に代入します。

```
FOR n := 0          TO 50   BY 1 DO
  SUM:= SUM + DATA[n] ;
END_FOR;
```

例 3) 配列変数 DATA[n] の要素の DATA[1] から DATA[50] の中の最大値と最小値を検索します。最大値を変数 MAX に代入し、最小値を変数 MIN に代入します。なお、DATA[n] の値は、0~1000 の範囲内とします。

```
MAX := 0;
MIN := 1000;
FOR n := 1          TO 50    BY 1 DO
  IF DATA[n] > MAX THEN
    MAX := DATA[n] ;
  END IF;
  IF DATA[n] < MIN THEN
    MIN := DATA[n] ;
  END IF;
END_FOR;
```

● WHILE 文

・ 概要：

ある条件式が真のあいだ、ある文を繰り返して実行します。

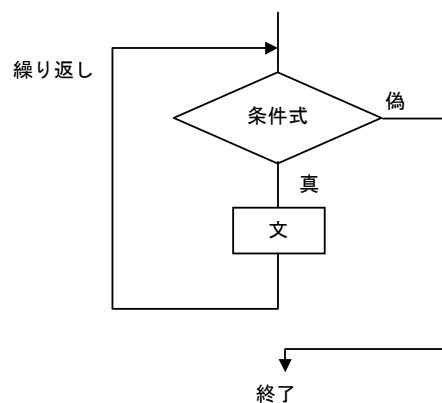
・ 予約語：

WHILE、DO、END_WHILE

・ 文の構造：

```
WHILE <条件式> DO
  <文>;
END_WHILE;
```

・ 処理フロー図：



・ 用途：

繰り返し回数がわかっていない（条件によって何回になるかわからない）場合で、ある条件式が成立している間、ある処理を繰り返すときに使用します。
条件式が成立しているときのみ、ある処理を実行するときに使用します（前判定の繰り返し）。

・ 説明：

<文>の実行前に、<条件式>が評価されます。

<条件式>が真の場合、<文>を実行します。その後、<条件式>が再度評価されます。これを繰り返し実行します。

<条件式>が偽の場合、<文>を実行せず、<条件式>の評価を終了します。

- ・ 留意事項：

- ・ WHILE は、必ず、END_WHILE と対で使用することが必要です。
- ・ <文>の実行前に、<条件式>が偽の場合、<文>を実行せず終了します。
- ・ <文>に、代入文、IF 文、CASE 文、FOR 文、WHILE 文、または REPEAT 文が可能です。
- ・ <文>の中で、複数の文を実行可能です。その場合、文と文の間を;で区切ります。
- ・ <条件式>には、式でなく、BOOL 型の変数のみを指定することもできます。

- ・ 例：

例 1) 7 の倍数で 1000 を超える数を算出し、変数 A に代入します。

```
A := 0;
WHILE A <= 1000 DO
  A := A+7;
END_WHILE;
```

例 2) 変数 X<3000 のあいだ、X の値を 2 倍し、その値を配列変数 DATA[1]に代入します。次にさらに X の値を 2 倍し、その値を配列変数 DATA[2]に代入します。これを繰り返します。

```
n := 1;
WHILE X < 3000 DO
  X:= X*2;
  DATA[n]:= X;
  n := n+1;
END_WHILE;
```

● REPEAT 文

- ・ 概要：

ある文を実行し、ある条件式が真になるまで、その文を繰り返します。

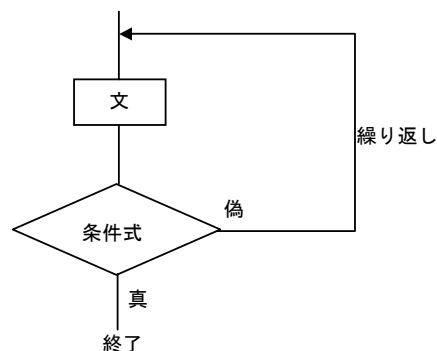
- ・ 予約語：

REPEAT、UNTIL、END_REPEAT

- ・ 文の構造：

```
REPEAT
  <文>;
UNTIL <条件式>
END_REPEAT;
```

- ・ 処理フロー図：



・用途：

繰り返し回数がわかっていない（条件によって何回になるかわからない）場合で、ある処理後の条件式が成立している間、その処理を繰り返すときに使用します。
ある処理を実行した結果に応じて、繰り返すかどうかを判断するときに使用します（後判定の繰り返し）。

・説明：

<文>が最初、無条件に、実行されます。その後、<条件式>が評価されます。
<条件式>が偽の場合、<文>を再度実行します。
<条件式>が真の場合、<文>を実行せず終了します。

・留意事項：

- ・REPEAT は、必ず、END_REPEAT と対で使用することが必要です。
- ・<文>の実行前に、<条件式>が真の場合でも、<文>が実行されます。
つまり、<文>は、すくなくとも 1 回は必ず実行されます。
- ・<文>に、代入文、IF 文、CASE 文、FOR 文、WHILE 文、または REPEAT 文が可能です。
- ・<文>の中で、複数の文を実行可能です。その場合、文と文の間を;で区切ります。
- ・<条件式>には、式でなく、BOOL 型の変数のみを指定することもできます。

・例：

例 1) 1 から 10 までの数値を加算し、変数 TOTAL に代入します。

```
A := 1;
TOTAL := 0;
REPEAT
    TOTAL := TOTAL + A;
    A := A+1;
UNTIL A>10
END_REPEAT;
```

● EXIT 文

・概要：

繰り返し文（FOR 文、WHILE 文、REPEAT 文）の中でのみ使用し、繰り返し文を強制的に終了させます。

IF 文の中で使用することで、ある条件が成立したとき、繰り返し文を強制的に終了させます。

・予約語：

EXIT

・文の構造（例：IF 文の中での使用）：

```
FOR (WHILE、REPEAT) 文
    ...
IF<条件式> THEN EXIT;
END_IF;
    ...
END_FOR (WHILE、REPEAT) ;
```

- ・ 用途：

終了条件が成立する前に、繰り返し処理を強制的に終了するときに、使用します。

- ・ 説明（例：IF 文の中での使用）：

<条件式>が真の場合、繰り返し文（FOR 文、WHILE 文、REPEAT 文）を強制終了し、EXIT 以降に記述されている文は実行されません。

注 1) <条件式>には、式でなく、BOOL 型の変数のみを指定することもできます。

注 2) <文>の実行前に、<条件式>が真の場合でも、<文>が実行されます。

- ・ 例：

例 1) 変数 n = 1 から 1 つおきに 50 まで、繰り返し、配列変数 DATA[n] に n を加算します。ただし、DATA[n] が 100 を超えたら、終了します。

```
FOR n := 1; TO 50 BY 1 DO
  DATA[n] := DATA[n] + n;
  IF DATA[n] > 100 THEN EXIT;
END_IF;
END_FOR;
```

● RETURN 文

- ・ 概要：

ST を使用するプログラムの種類によって、以下のようになります。

<ST プログラム>

途中で、ST プログラムタスクを強制終了して、次のタスクを実行します。

<SFC プログラム>

途中で、アクションプログラムを強制終了して、次のアクションプログラムまたはトランジションプログラムを実行します。

<FB のインスタンス内で使用する ST>

途中で、ST 言語の本ファンクションブロックを強制終了して、プログラム上の本ファンクションブロック呼び出し元インスタンスの、次の命令を実行します。

- ・ 予約語：

RETURN

- ・ 文の構造：

```
RETURN;
```

- ・ 用途：

実行中の ST タスク、SFC プログラムのアクションプログラムおよびファンクションブロックを強制終了するときに使用します。

● ファンクションブロック呼び出し文

・ 概要：

ファンクションブロック定義を呼び出します。

・ 予約語：なし

・ 文の構造：

インスタンス名*1の後に、（ ）内に、引数指定（指定した変数の値を、呼び出し先のファンクションブロックの入力変数に渡す指定）と戻り値指定（呼び出し先のファンクションブロックの出力変数の値を、指定した変数に受ける指定）を記述します。

以下の①または②のいずれかの記述方式が可能です。

*1：データ型がファンクションブロックの任意の内部変数名（ファンクションブロックのインスタンス内で使用する ST の場合）、またはグローバル変数名（ST タスク、または SFC のアクションプログラムで使用する ST の場合）です。

・ 記述方式①：

引数指定（呼び出し先ファンクションブロック定義の変数名）と戻り値指定を、いずれも記述する場合

A(B:=C, ... ,D=>E)

A：インスタンス名

B：呼び出し先ファンクションブロック定義の入力変数名

C：使用する ST プログラムにより、以下のように異なります。

・ 自身の入力変数または定数

（ファンクションブロックのインスタンス内で使用する ST の場合）

・ グローバル変数／ローカル変数名

（ST タスク、または SFC のアクションプログラムで使用する ST の場合）

D：呼び出し先ファンクションブロック定義の出力変数名または定数

E：使用する ST プログラムにより、以下のように異なります。

・ 自身の出力変数または定数

（ファンクションブロックのインスタンス内で使用する ST の場合）

・ グローバル変数／ローカル変数名

（ST タスク、または SFC のアクションプログラムで使用する ST の場合）

注：上記“B:=C”は、すべてを“,”で区切って記述します。

上記“D=>E”は、必要数のみを“,”で区切って記述します。

・記述方式②：

引数指定（呼び出し先ファンクションブロック定義の変数名）を省略し、戻り値指定のみを記述する場合

A(C, ..., E)

A：インスタンス名

B：省略します。（呼び出し先ファンクションブロック定義の入力変数名）

C：使用する ST プログラムにより、以下のように異なります。

- ・自身の入力変数または定数

（ファンクションブロックのインスタンス内で使用する ST の場合）

- ・グローバル変数／ローカル変数名

（ST タスク、または SFC のアクションプログラムで使用する ST の場合）

D：省略します。（呼び出し先ファンクションブロック定義の出力変数名または定数）

E：使用する ST プログラムにより、以下のように異なります。

- ・自身の出力変数または定数

（ファンクションブロックのインスタンス内で使用する ST の場合）

- ・グローバル変数／ローカル変数名

（ST タスク、または SFC のアクションプログラムで使用する ST の場合）

注：上記のように、B、D を省略した場合は、C を B へ、その変数テーブル上での登録順で自動的に渡します。逆に、D から、その変数テーブル上での登録順で、自動的に E に受け取ります。

・用途：

ST 言語からファンクションブロック定義（ST 言語またはラダー言語）を呼び出すときに使用します。

・説明／留意事項：

1. 変数テーブル上の内部変数、またはグローバル変数に、以下のインスタンスを登録します。

内部変数要素	内容	例
名称	任意のインスタンス名	Calcu_execute
データ型	ファンクションブロック	FUNCTION BLOCK
FB 定義	呼び出し先ファンクションブロック定義を選択	Calculation

2. ST 言語プログラム上に、手順 1 で登録したインスタンス名（例：Calcu_execute）の後の（）内に、自身の変数と呼び出す変数との間での値の受け渡しを、例えば、以下のように、記述し、文として、最後;で終了します。

Calcu_execute (A:=B,C=>D) ;

B の値を A に渡します。同時に、C の値を D で受けます。

A：呼び出し先ファンクションブロック定義の入力変数名

B：使用する ST プログラムにより、以下のように異なります。

- ・自身の入力変数または定数

（ファンクションブロックのインスタンス内で使用する ST の場合）

- ・グローバル変数／ローカル変数名

（ST タスク、または SFC のアクションプログラムで使用する ST の場合）

C：呼び出し先ファンクションブロック定義の出力変数名または定数

D：使用する ST プログラムにより、以下のように異なります。

- ・自身の出力変数または定数

（ファンクションブロックのインスタンス内で使用する ST の場合）

- ・グローバル変数／ローカル変数名

（ST タスク、または SFC のアクションプログラムで使用する ST の場合）

・詳細説明のための例

実際に ST プログラムを使って、ファンクションブロックを呼び出す使い方を、以下の①～②の例で説明します。

例①：

ファンクションブロックを呼び出す例を、呼び出し元の種類（ST タスク、SFC、ファンクションブロック）ごとに例を説明します。

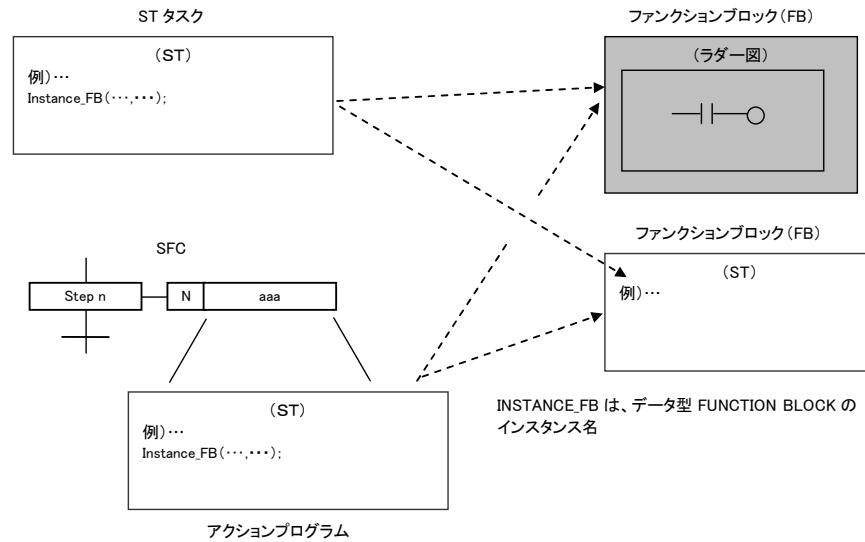
条件：

ファンクションブロックを呼び出します。

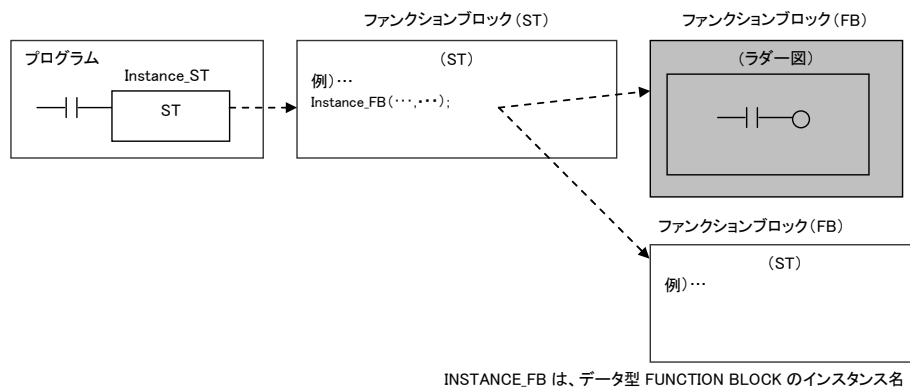
呼び出すファンクションブロックは、ラダー言語または ST 言語。

[呼び出し内容]

a) ST タスク、SFC プログラムから、ファンクションブロックを呼び出す場合



b) ファンクションブロックから呼び出す場合



[変数設定]

・ ST プログラム、SFC プログラム（呼び出し元）の変数設定

ST プログラム、SFC プログラムは、以下の変数を持ち、ファンクションブロック（呼び出し先）と以下の受け渡しをします。

ST タスク／SFC プログラムの変数名	ファンクションブロック（呼び出し先）の変数へ渡す／から受ける
IN1	FB_IN1（入力変数）へ渡す
IN2	FB_IN2（入力変数）へ渡す
IN3	FB_IN3（入力変数）へ渡す
OUT1	FB_OUT1（出力変数）から受ける
OUT2	FB_OUT2（出力変数）から受ける
OUT3	FB_OUT3（出力変数）から受ける
A 注) データ型: Bool	EN（内部変数）へ渡す
B 注) データ型: Bool	ENO（内部変数）から受ける
Instance_FB 注) データ型: FUNCTION BLOCK	呼び出すファンクションブロック定義: ファンクションブロック

- ・ファンクションブロック（呼び出し元）の変数設定

ファンクションブロック（呼び出し元）は、以下の変数を持ち、ファンクションブロック（呼び出し先）と以下の受け渡しをします。

変数の種類	ファンクションブロック（呼び出し元）の変数名	ファンクションブロック（呼び出し先）の変数へ渡す／から受ける
入力変数	IN1	FB2_IN1 へ渡す
	IN2	FB2_IN2 へ渡す
	IN3	FB2_IN3 へ渡す
出力変数	OUT1	FB2_OUT1 から受ける
	OUT2	FB2_OUT2 から受ける
	OUT3	FB2_OUT3 から受ける
内部変数	A 注) データ型: Bool	EN へ渡す
	B 注) データ型: Bool	ENO から受ける
内部変数 (インスタンス)	Instance_FB 注) データ型: FUNCTION BLOCK	呼び出すファンクションブロック定義: ファンクションブロック 2

- ・ファンクションブロック（呼び出し先）の変数設定

ファンクションブロック（呼び出し先）は、以下の変数を持ち、呼び出し元（ST プログラム／SFC プログラム／呼び出し元のファンクションブロック）と以下の受け渡しをします。

変数の類	ファンクションブロック（呼び出し先）の変数名	呼び出し元の変数から受ける／へ渡す
入力変数	FB_IN1	IN1 から受ける
	FB_IN2	IN2 から受ける
	FB_IN3	IN3 から受ける
出力変数	FB_OUT1	OUT1 へ渡す
	FB_OUT2	OUT2 へ渡す
	FB_OUT3	OUT3 へ渡す

[記述例]

- ・記述方式①の例:

Instance_FB(EN:=A,FB_IN1:=IN1,FB_IN2:=IN2,FB_IN3:= IN3,FB_OUT1=>OUT1,FB_OUT2=> OUT2,FB_OUT3=> OUT3,ENO=>B)

- ・各引数、戻り値の順番は、順不同でかまいません。
- ・EN を除くすべての入力変数分の引数が最低限必要です。
- ・出力変数については、実際に使用しないデータは省略可能です。
- ・同一のファンクションブロック呼び出し文内での、記述方式②の記述方法との混合使用は不可です。

- ・その他の記述例:

- ・EN を記述しない場合

Instance_FB(FB_IN1:=IN1,FB_IN2:= IN2,FB_IN3:= IN3,FB_OUT1=>OUT1,FB_OUT2=> OUT2,FB_OUT3=> OUT3,ENO=>B)

- ・EN、ENO を記述しない場合

Instance_FB(FB_IN1:=IN1,FB_IN2:= IN2,FB_IN3:= IN3,FB_OUT1=>OUT1,FB_OUT2=> OUT2,FB_OUT3=> OUT3)

- ・ENO を記述しない場合

Instance_FB(EN:=A,FB_IN1:=IN1,FB_IN2:=IN2,FB_IN3:=IN3,FB_OUT1=>OUT1,FB_OUT2=>OUT2,FB_OUT3=>OUT3)

- ・ FB_OUT2 のデータが不要な場合

```
Instance_FB(EN:=A,FB_IN1:=IN1,FB_IN2:=IN2,FB_IN3:=IN3,FB_OUT1=>OUT1,FB_OUT3=>OUT3,ENO=>B)
Instance_FB(FB_IN1:=IN1,FB_IN2:=IN2,FB_IN3:=IN3,FB_OUT1=>OUT1,FB_OUT3=>OUT3)
```

- ・ 順不同とした場合

```
Instance_FB(EN:=A,FB_IN1:=IN1,FB_OUT1=>OUT1,FB_IN2:=IN2,FB_OUT2=>OUT2,FB_IN3:=IN3,FB_OUT3=>OUT3,ENO=>B)
```

- ・ 記述方式②の例：

親インスタンスのパラメータ変数（定数値含む）のみを記述する

```
Instance_FB(IN1, IN2, IN3, OUT1, OUT2, OUT3)
```

```
Instance_FB(IN1, IN2, IN3, OUT1)
```

- ・ 各引数、戻り値の順番は固定です。

入力変数 1, 入力変数 2, ..., 出力変数 1, 出力変数 2, ...

- ・ EN を除くすべての入力変数分の引数が最低限必要です。

- ・ 出力変数については、実際に使用しないデータは省略可能です。ただし、途中のデータの省略は不可です。

例) Instance_FB(IN1, IN2, IN3, OUT1, OUT3)

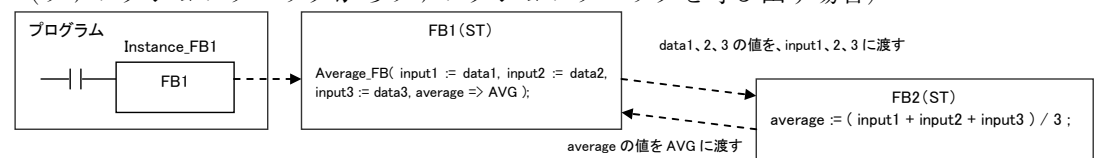
この場合、最後の OUT3 は、FB_OUT2 からの戻り値と判断されます。

- ・ EN、ENO に関するデータは、引数、戻り値に記述不可です。
- ・ 同一のファンクションブロック呼び出し文内での、記述方式①の記述方法との混合使用は不可です。

例②：

以下のように、平均値を計算するファンクションブロック 2 を呼び出します。

（ファンクションブロックからファンクションブロックを呼び出す場合）



Average_FB は、データ型 FUNCTION BLOCK のインスタンス名

ファンクションブロック 1

- ・ 変数テーブル：

変数種類	変数名	データ型	ファンクションブロック 2 の変数へ渡す／から受ける
入力変数	EN	BOOL	
入力変数	data1	INT	input1 へ渡す
入力変数	data2	INT	input2 へ渡す
入力変数	data3	INT	input3 へ渡す
入力変数	bCheck	BOOL	
出力変数	ENO	BOOL	
出力変数	AVG	INT	average から受ける
内部変数	Average_FB	FUNCTION BLOCK 呼び出すファンクション ブロック定義: ファンクシ ョンブロック 2	

5-5 各構文の説明

5-5-2 制御文

- ST 言語アルゴリズム：

bCheck が真であれば、平均値を計算するファンクションブロック 2 を呼び出します。 data1、2、3 の値を、各々、ファンクションブロック 2 の input1、2、3 へ渡します。その演算結果 average を AVG で受けます。

注：以下は記述方式 A で、ファンクションブロック呼び出し文 Average_FB () を記述しています。

```
IF bCheck = TRUE THEN
  Average_FB( input1 := data1, input2 := data2, input3 := data3, average => AVG );
ELSE
  RETURN;
END_IF;
```

ファンクションブロック 2

- 変数テーブル：

変数種類	変数名	データ型	ファンクションブロック 1 の変数から受ける/へ渡す
入力変数	EN	BOOL	
入力変数	input1	INT	data1 から受ける
入力変数	input2	INT	data2 から受ける
入力変数	input3	INT	data3 から受ける
出力変数	ENO	BOOL	
出力変数	average	INT	AVG へ渡す

- ST 言語アルゴリズム：

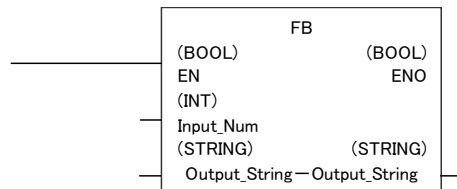
input1、input2、input3 の平均値の計算をし、average に格納します。

```
average := ( input1 + input2 + input3 ) / 3 ;
```

5-6 ST 言語プログラム使用例

5-6-1 ファンクションブロックによる ST 言語プログラム使用例

■ 例 1：整数値を BCD データに変換して文字列出力するプログラム



[入力変数]

INT Input_Num;

[入出力変数]

STRING Output_String;

[内部変数]

WORD Num_BCD;

(*入力パラメータ Input_BCD(BCD データ)のチェックを行う*)

IF (Input_Num>=0 & Input_Num<=9999) THEN

ENO:=true;

ELSE

ENO:=false;

RETURN;

END_IF;

Num_BCD := INT_TO_BCD_WORD(Input_Num);

(* 例えば Num の値が 100(16#0064)のとき、BCD 値 0100 に変換する *)

Output_String := WORD_TO_STRING(Num_BCD);

(* BCD 値 0100 を文字列に変換する *)

5-7 制限事項

5-7-1 制限事項

- ・ネスト数について
 - ・IF 文、CASE 文、FOR 文、WHILE 文、REPEAT 文のネスト数に、制限はありません。
- ・データ型の制限事項
 - ・整数値は、WORD、DWORD、INT、DINT、UINT、UDINT、ULINT のデータ型の変数に対してのみ割付可能です。例：A が INT 型の場合、A:=1;とすることが可能です。そうでない場合、文法エラーとなります。例：A が INT 型の場合、A:=2.5;とすると、文法エラーとなります。
 - ・実数値（浮動小数点データ）は、REAL、LREAL のデータ型の変数に対してのみ割付可能です。例：A が REAL 型の場合、A:=1.5;とすることが可能です。そうでない場合、文法エラーとなります。例：A が REAL 型の場合、A:=2;とすると、文法エラーとなります。A:=2.0;としてください。
 - ・接点（TRUE、FALSE）は、BOOL のデータ型の変数に対してのみ割付可能です。例：A が BOOL 型の場合、A:=FALSE;とすることが可能です。そうでない場合、文法エラーとなります。例：A が INT 型の場合、A:=FALSE;とすると、文法エラーとなります。
 - ・ST 言語内で、データ型はすべて整合していないといけません。
例：A、B、C が INT 型の場合、A:=B+C;は可能です。
これに対して、A、B が INT 型で C が REAL 型の場合、または A、B が INT 型で C が LINT 型の場合、A:=B+C;は文法エラーとなります。
 - ・ST 言語では、P_CY、P_EQ、P_ER、P_N、P_GE、P_GT、P_LE、P_LT、P_NE、P_OF、P_UF を使用することはできません。
- ・モニタに関する制限事項
 - ・タイマ関数のモニタ時に、以下の制限事項があります。
0.1ms タイマまたは 0.01ms タイマで TIMER 型変数を使用している場合、ST モニタビューではこのタイマ関数上の TIMER 型変数の現在値は表示されません。現在値は「-」と表示されます。
これに伴い、この TIMER 型変数の現在値を ST エディタ内の他の箇所で使用するか、または別変数に代入して使用している場合、使用箇所での現在値には不定な値が表示されますのでご注意ください。
 - ・変数名に全角文字を使用する場合、以下の制限事項があります。
変数名に全角（2 バイト）文字を使用する場合、変数と演算子の間に半角スペースを入れてください。入れない場合、変数の現在値を正しくモニタリングできないことがあります。

5-7-2 Q&A (参考)

●16 進数値の書き方は？

16#123F のように数値の前に 16# を付加します。

16 進表記の他にも、8# (8 進)、2# (2 進) が利用できます。進数指定を省略すると、10 進数として解釈されます。

●FOR の回数は？

下記の例では FOR 文の中は 101 回実行されます。i が 101 に等しくなったときにループ処理が終了します。

```
FOR i:=0 TO 100 BY 1 DO  
  a:=a+1;  
END_FOR;
```

●配列添字を超えた場合の動作は？

要素数 10 の配列変数 INT[10] に対して、下記のような文を記述した場合など、異常を検知できません。実行時の動作は不定となります。

```
i:=15;  
INT[i] :=10;
```

●ST エディタ上で記述した変数は、変数テーブル上に自動登録されないのですか？
自動登録されません。変数テーブル上で手動登録してから、利用してください。

●ラダープログラムの命令を直接呼べないの？
呼ぶことはできません。

5

S
T
言
語
の
仕
様

ストラクチャードテキスト編

第6章 ST プログラムの 作成

6-1 操作方法

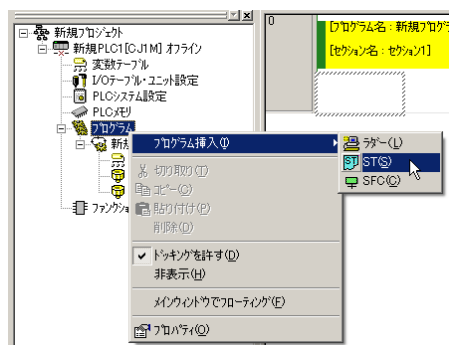
ST プログラムの作成方法を説明します。ファンクションブロックを ST 言語で作成する方法については、ファンクションブロック編の「3 章 ファンクションブロックの作成」を参照してください。

6-1-1 プロジェクトを作成する

- 1 CX-Programmer を起動し、[ファイル] メニューから [新規作成] を選択します。
- 2 [PLC 機種変更] ダイアログボックスにて、[PLC 機種] リストから、ST プログラムが使用可能な PLC 機種を選択します。
ST プログラムが使用可能な PLC の機種については、ストラクチャードテキスト編の 4-2-1 項を参照してください。
- 3 [設定] ボタンを押して CPU タイプを選択します。
それ以外の設定は、CX-Programmer オペレーションマニュアル (No.SBCA-337) を参照してください。

6-1-2 ST プログラムの新規作成 (挿入)

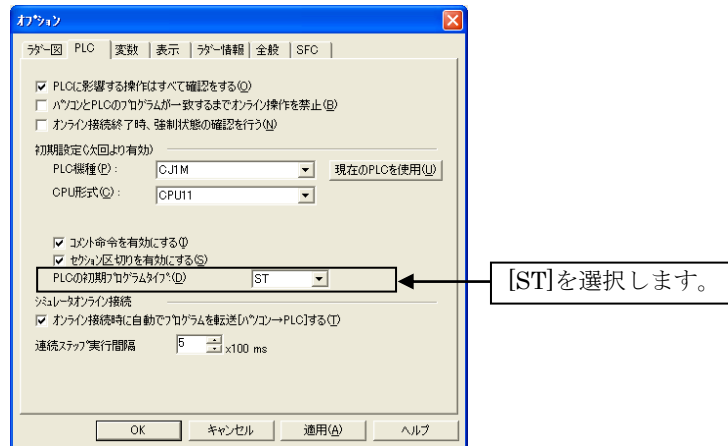
プロジェクトに ST プログラムを作成する場合、以下の操作を実行します。
プロジェクトワークスペース上の [プログラム] アイテムを右クリックし、ポップアップメニューから [プログラム挿入] | [ST] を選択します。



プロジェクトワークスペースに ST プログラムが挿入されると、ワークスペースの右側に ST ダイアグラムエディタが表示されます。

注： ラダー、SFC もプログラムを作成することができます。プロジェクトワークスペース上の [プログラム] アイテムを右クリックし、ポップアップメニューから [プログラム挿入] | [ラダー] または [プログラム挿入] | [SFC] を選択します。
ラダーについては、CX-Programmer のオペレーションマニュアル (SBCA-337) を参照してください。
SFC については、CX-Programmer のオペレーションマニュアル・SFC 編 (SBCA-348) を参照してください。

注： 新規プロジェクトを作成したとき、ST プログラムを初期プログラムに設定することができます。[ツール] | [オプション] の [PLC タブ] にて設定します。



6-1-3 ST プログラムのタスクへの割り付け

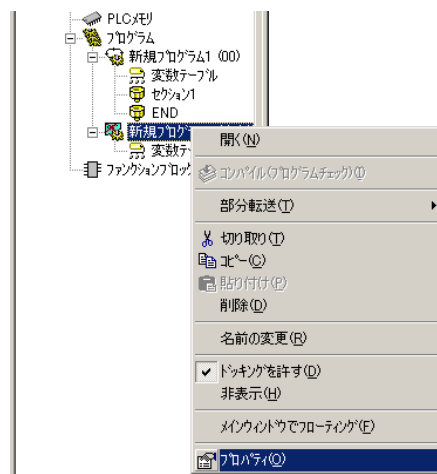
プロジェクトに挿入した ST プログラムは、実行単位としてのタスクに割り付ける必要があります。

タスクに割り付けられていないプログラムは、プロジェクトワークスペース上のプログラムアイコンにチェックマークがついています。

注： 以下のプログラムのタスクへの割り付け操作は、プログラム内容作成後でもかまいません。ただし、ユーザプログラムとして PLC へ転送する前には必ず割り付けてください。

タスクに割り付けるには、以下の操作を実行します。

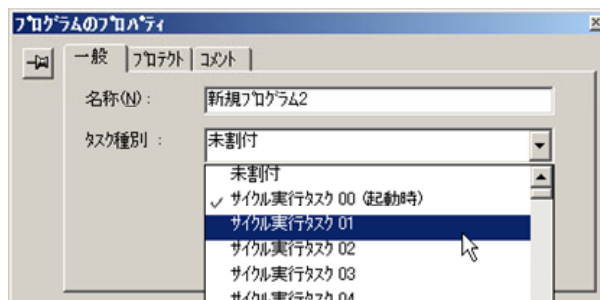
- 1 プロジェクトワークスペース上から、挿入した ST プログラムアイテムを右クリックし、ポップアップメニューから [プロパティ] を選択します。



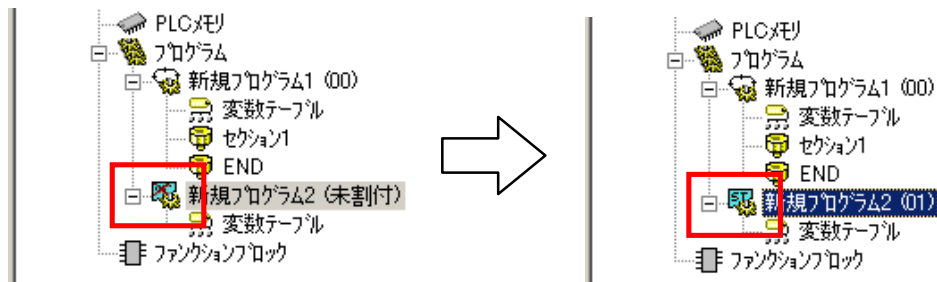
- 2 表示された [プログラムのプロパティ] ダイアログの [一般] タブ | [タスク種別] リストから、タスクを選択します。
プログラム名称を設定する場合は、ここで [名称] テキストボックスにプログラム名称を入力します。

6-1 操作方法

6-1-3 ST プログラムのタスクへの割り付け



- 3 クローズボタンで「プログラムのプロパティ」ダイアログを閉じます。



- 4 プログラムがタスクに割り付けられると、プログラムツリー内の ST プログラムアイテムのアイコンからチェックマークの表示が消えます。また、プログラム名の () 内の表示は、割り付けられたタスク番号を示しています。

6-1-4 ST プログラムを作成する

ST プログラムの内容の作成方法には、以下の 2 通りがあります。

- ・変数の登録の後、ST 言語を入力する方法
- ・ST 言語を入力する都度、変数を登録する方法

■ 変数の登録の後、ST 言語を入力する場合

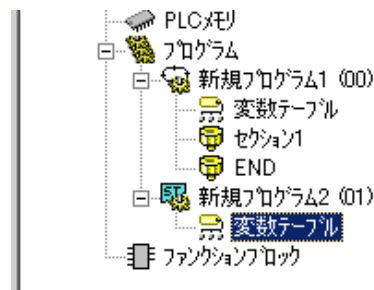
変数の種類には、グローバル変数とローカル変数の 2 種類があります。

ここでは、ローカル変数の設定方法を説明します。

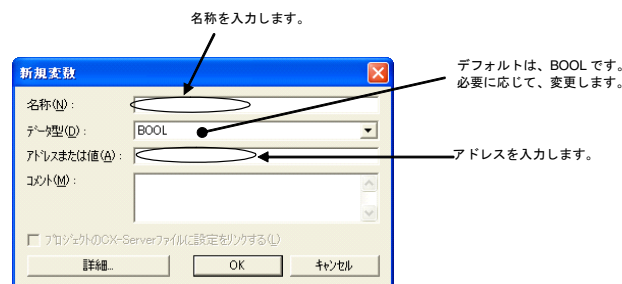
グローバル変数についての設定方法は、CX-Programmer オペレーションマニュアル (No.SBCA-337) を参照してください。

1) 変数の登録 (ローカルアドレスの場合)

1. プロジェクトワークスペース上から、挿入した ST プログラムアイテムの[変数テーブル]をダブルクリックします。



2. 変数テーブルが表示されますので、右クリックし、ポップアップメニューから「変数挿入」を選択します。
または「挿入」メニューから「変数」を選択します。
3. 以下の[新規変数]ダイアログボックスが表示されます。
以下の項目を設定して、「OK」ボタンをクリックします。
 - ・名称：変数名を入力します。
 - ・データ型：データタイプを選択します。
 - ・アドレスまたは値：アドレスを入力します。

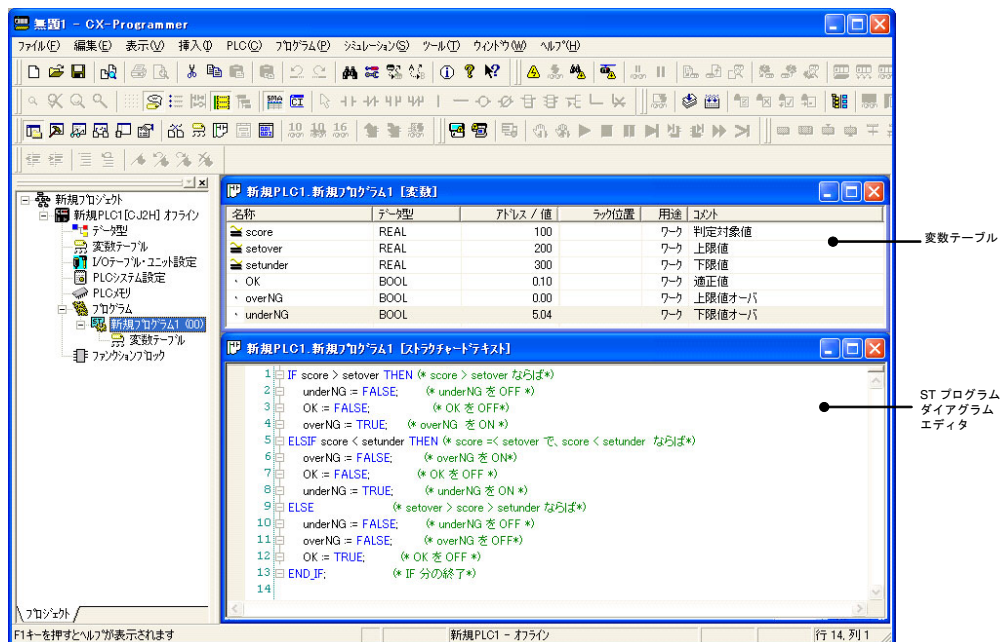


4. 「変数テーブル」に設定した変数が登録されます。

参 考

アドレスを指定しないで変数を登録すると、アドレスを自動割付で設定することができます。アドレスの自動割付の設定についての詳細は、CX-Programmer のオペレーションマニュアル (SBCA-337) を参照してください。

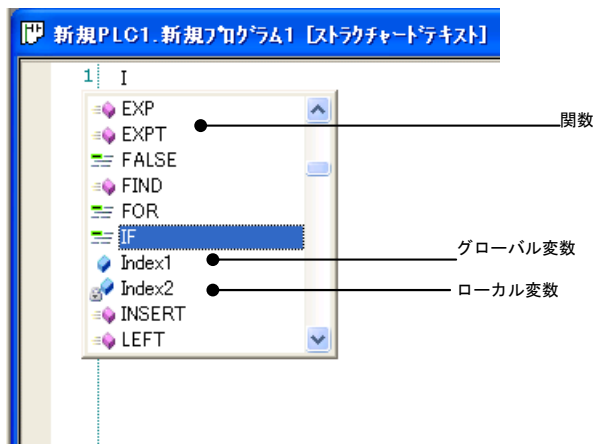
2) ST プログラム作成



- 注 1： ST 入力画面にテキストを入力または貼り付けると、文のキーワード(予約語)は青字、コメントは緑字、エラーは赤字、文字列は茶色、その他は黒字で、自動的に表示されます。
- 注 2： フォントサイズ、表示色を変更したい場合は、[ツール] メニューから [オプション] を選択して、[表示] タブで、[ST フォント] ボタンをクリックします。フォント名、フォントサイズ(デフォルトは9ポイント)を変更できます。表示色を変更した場合は、同じく [表示] タブで項目を選択し表示色を変更します。
- 注 3： ST 言語の仕様について詳細は、ストラクチャードテキスト編 第5章「ST 言語の仕様」を参照してください。
- 注 4： 汎用のテキストエディタで作成した ST 言語のテキストを [編集] メニューから [貼り付け] を選択して貼り付けることもできます。
- 注 5： ST 関数および ST 構文要素は、自動的に大文字変換して、表示します。

● 関数および変数の入力

1. ST プログラムエディタ上で使用する関数または登録済変数の先頭文字を入力することで、キーワードリストが表示されます。



各キーワードが関数か変数かは、各キーワード左側のアイコンで確認できます。

アイコン	キーワードの種類
	関数
	制御構文要素
	グローバル変数
	ローカル変数 ファンクションブロック変数

2. リスト上で入力する関数または変数を選択し、[Enter] キー、[Space] キーまたは [Tab] キーを押します。選択が確定し、ST プログラムエディタに反映されます。

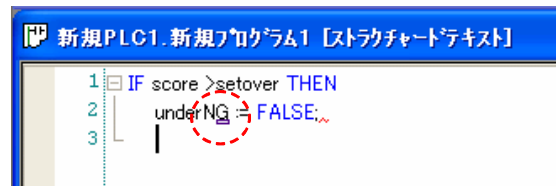
注1：キーワードリスト上からの選択をやめる場合には、[Esc] キーを押します。

注2：キーワードリストから選択せず、ST プログラムエディタ上で直接入力することも可能です。

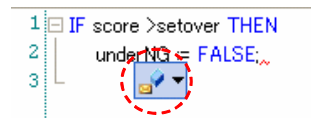
■ ST 言語を入力する都度、変数を登録する場合

ST 言語を先に記述したとき、入力中に変数テーブルへの入力を促すダイアログボックスは表示されません。

1. 各行入力完了後、[Enter] キーを押すと、未登録変数の下に 2 重線が表示されます。



2. 2 重線にマウスカーソルを当てると、ボタンが表示されます。



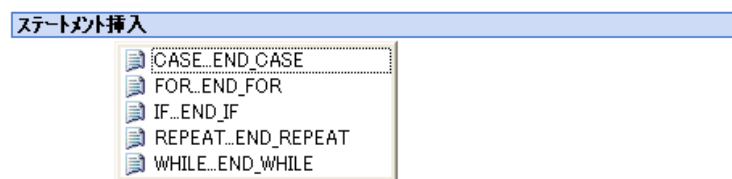
3. このボタンを押すと、新規変数登録ダイアログが表示されますので、各項目を設定して、[OK] ボタンをクリックします。

■ ST エディタでの入力アシスト機能

- 1) 制御構文ステートメントの入力

以下の 2 つの方法で、簡単に制御構文ステートメントフレームを入力できます。

1. ポップアップメニュー「制御ステートメントの挿入」を選択し、表示される以下のリストから選択します。



2. キーワードリストから制御構文先頭要素を選択入力後、[Tab] キーを押します。

キーワード	制御構文ステートメントフレーム
IF	IF expression THEN ELSE END_IF;
FOR	FOR variable := expression1 TO expression2 BY expression3 DO END_FOR;
CASE	CASE expression OF ELSE END_CASE;
REPEAT	REPEAT UNTIL expression END_REPEAT;
WHILE	WHILE expression DO END_WHILE;

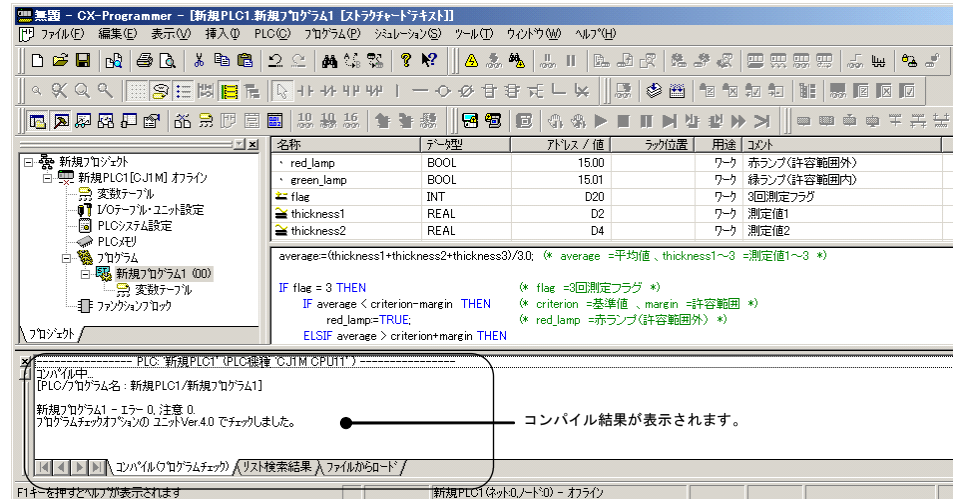
注：各[expression],[variable]には、制御構文実行条件を入力する必要があります。

- キーワードリストでの選択キーワード内容をツールチップで表示しますので、キーワード関数の動作概要や変数のコメントなどを確認しながら、選択可能です。
- 各行のインデントをメニューまたはツールボタンにより、増減させることができます。
- 選択行のコメント化切り替えを、メニューまたはツールボタンにより実施できます。

6-1-5 ST プログラムをコンパイル (プログラムチェック) する

ST プログラムだけをコンパイルして、プログラムチェックをかけることができます。以下の操作を実行します。

1. ST プログラムを選択し、右クリックしてポップアップメニューの「コンパイル (プログラムチェック)」を選択、または「Ctrl」+「F7」キーを押します。
2. コンパイルが実行され、アウトプットウィンドウの「コンパイル」タブが自動的に表示されて、プログラムチェック結果が表示されます。



6-1-6 プログラムを実機 PLC へダウンロード/アップロードする

ST プログラムを使用したプログラムを作成後、CX-Programmer を実機 PLC とオンライン接続し、実機 PLC へプログラムをダウンロードします。また、逆にプログラムを実機 PLC からアップロードします。

ただし、タスク単位でダウンロード/アップロードすることはできません。

6-1-7 ST プログラムの照合

作業中の ST プログラムと実機 PLC または他のプロジェクトファイル間で、ST プログラムが同じかどうかの照合ができます。

プログラムの照合についての詳細は、CX-Programmer オペレーションマニュアル (SBCA-337) を参照してください。

6-1-8 ST プログラムをモニタ/デバッグする

■ ST プログラムの変数モニタ

ST プログラムをモニタすることができます。

左側（「ST プログラム画面」と呼びます）に、ST プログラムが表示されます。

右側（「ST 変数モニタ画面」と呼びます）に、ST で使用している変数の値が表示されます。

このとき、以下のように、変数モニタ、現在値変更、強制セット/リセット、ウォッチウィンドウへのコピー/ペーストが可能です。

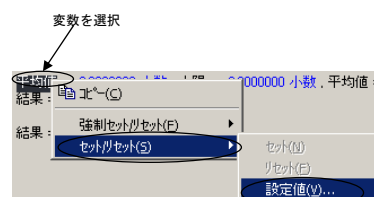
・変数モニタ

ST 変数モニタ画面内の変数の値が青字で表示されます。

注：0.1ms タイマまたは 0.01ms タイマで TIMER 型変数を使用している場合、この TIMER 型変数の現在値は「-」と表示されます。詳細は、「5-7 制限事項」を参照してください。

・現在値変更

ST 変数モニタ画面内の変数自体を選択して（反転させて）、右クリックしてポップアップメニューから [セット/リセット] | [設定値] を選択します。



[新しい値を設定] ダイアログボックスが表示されます。

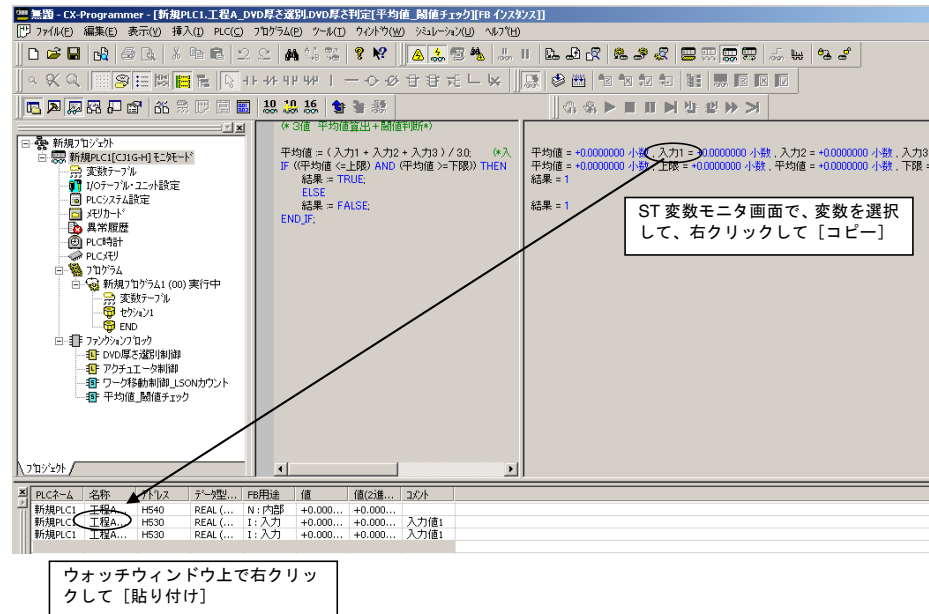
[値] を入力します。

・強制セット/リセット

ST 変数モニタ画面内の変数自体を選択して（反転させて）、右クリックしてポップアップメニューから [強制セット/リセット] | [セット] / [リセット] / [キャンセル] / [すべて解除] を選択します。

・ウォッチウィンドウへのコピー/ペースト

1. ST 変数モニタ画面内の変数自体を選択して（反転させて）、右クリックしてポップアップメニューから「コピー」を選択します。
2. ウォッチウィンドウ内にて右クリックし「貼り付け」を選択します。



■ ST プログラムのシミュレーション機能

ST プログラムをシミュレータに接続してモニタすることが可能です。

6-1-9 ST プログラムのオンラインエディット

PLC（CPU ユニット）が運転中の場合でも、ST プログラムの内容を変更することができます。

これにより、24 時間稼動など、運転を停止できない装置においても、ST プログラム自体のデバッグ/変更が可能となります。

PLC が運転モード以外のときに、ST プログラムをオンラインエディットできます。ただし、本機能は、シミュレータで使用することはできません。

■ オンラインエディットを開始する

1. モニタリングを開始します。
2. プロジェクトワークスペース上から、編集する ST プログラムを選択してプログラムビューに表示させます。
3. [プログラム] | [オンラインエディット] | [開始] を選択します。ST プログラムが編集可能な状態になります。
4. ST プログラムの編集を開始してください。

■ 変更を転送する

1. 編集終了後、[プログラム] | [オンラインエディット] | [変更を転送] を選択します。[変更を転送] ダイアログが表示されます。
2. 転送モードを選択し、[OK] をクリックします。編集した ST プログラムが PLC に転送されます。
転送モードの詳細は、後述の「●転送モードについて」、「●転送モードの使い分けについて」を参照してください。
3. 転送終了後、ST プログラムは自動的に編集不可能な状態に戻ります。追加で編集作業を行う場合は、再度オンラインエディットの開始から始めてください。

■ キャンセル

1. 編集した ST プログラムの内容を破棄する場合は、[プログラム] | [オンラインエディット] | [取消] を選択します。編集した ST プログラムは PLC に転送されず、編集前の状態に戻ります。

■ 転送モードについて

● 標準モード

ST プログラムの ST ソースコードとオブジェクトコードをともに PLC（CPU ユニット）に転送します。

「標準モード」では、転送するデータ量により、転送に時間がかかることがあります。その間は次の編集および転送操作を行うことができません。

・ クイックモード

ST プログラムのオブジェクトコードのみを PLC（CPU ユニット）に転送します。ST ソースコードは転送しません。

「クイックモード」では、オブジェクトコードのみを転送するため、「標準モード」より高速で転送が可能です。

「クイックモード」でオブジェクトコードを転送した後は、①「ST ソース転送」を実行するとき、または②オフラインに切り替えるときに、ST プログラムの ST

ソースコードを PLC (CPU ユニット) に転送します。変更を転送後、オフライン状態に切り替えるまで、画面下に、「ST ソースコードが転送されていないこと」を示す黄色のマークが表示されます。ST ソースコード転送終了後、黄色のマークが消えます。

■ 転送モードの使い分けについて

基本的に、頻繁に連続してオンラインエディットをするとき以外は、「標準モード」で転送をしてください。転送に時間がかかるときは、できるだけ PLC との通信速度を上げて、転送処理を行ってください。

それでも転送時間がかかりすぎて、デバッグに支障がある場合（デバッグ時に頻繁に連続してオンラインエディットをするときなど）は、例外的に、以下の「クイックモード（ST ソースコードの転送なし）の制約」を十分ご理解いただいた上で、「クイックモード」で転送してください。

お願い

クイックモード（ST ソースコードの転送なし）の制約：

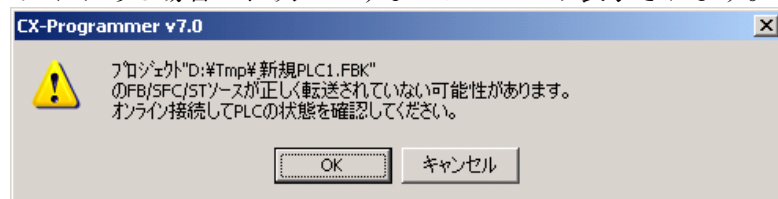
ST プログラムの ST ソースコードを転送していない状態では、**CX-Programmer** は、正しいプログラムを次にアップロードすることができません。

したがって、オンラインエディット時に「クイックモード」で ST プログラムの変更内容を転送した後、オフラインに切り替える前＝ST ソースコード転送終了前）に、万が一パソコン（**CX-Programmer**）のダウン、または PLC 本体の電源断が発生した場合、**CX-Programmer** がプログラムをアップロードできない状態に陥る可能性がありますので、ご注意ください（*1）。

*1: なお、その場合、以下に示す操作によって、ST ソースコードを再度転送することができる場合があります。

● バックアッププロジェクトからソースコードを転送する方法

1. **CX-Programmer** を起動します。
2. ST ソースコードの転送を以前クイックモードで行い、転送に失敗したプロジェクトがある場合は、次のようなメッセージが表示されます。



3. [OK]ボタンをクリックすると、以前クイックモードで転送を行ったバックアッププロジェクトが起動します。
4. 以前クイックモードで転送した PLC へオンライン接続します。次のようなメッセージが表示されます。



5. [はい]ボタンをクリックします。
 運転モード以外の場合、プロジェクトと PLC 間でプログラムの照合を行い、照合が一致したら ST ソースコードの転送を行います。
 運転モードの場合は、動作モードを運転モード以外に変更してから、メニューより ST ソースコードの転送を行ってください。

● 手動で ST ソースコードを転送する方法

1. CX-Programmer を起動し、ST ソースコードを転送するプロジェクトファイルを開きます。
2. 以前クイック転送を実行した PLC にオンライン接続します。CX-Programmer のステータスバーに、[ソース不一致]と表示され、黄色点滅します。
3. [プログラム] | [オンラインエディット] | [SFC/ST ソース転送] を選択します。ST ソースコードの転送ダイアログボックスが表示されます。
4. [OK] ボタンをクリックします。
パソコン内に自動的にバックアップされていた ST ソースコードと、実機 PLC 内のオブジェクトコードを照合し、一致した場合、ST ソースコードの転送を行うことができます。

参考

通常、**CX-Programmer** はプログラム転送前に、プログラムコード（ST ソースコード）を **CPU ユニット** が実行可能なコード（オブジェクトコード）に変換（コンパイル）して、その両方を転送します。

なお、**CPU ユニット** は、プログラムの ST ソースコードおよびオブジェクトコードをユーザメモリおよび内蔵フラッシュメモリに保存します。この両方のコードが保存されている場合のみ、**CX-Programmer** は **CPU ユニット** からの転送操作によって、転送したプログラムを復元することができます。

■ ST プログラムのオンラインエディットにおける制限事項

ST プログラムのオンラインエディットにおける制限事項は以下のとおりです。

- ・ CJ2 シリーズについては、1 回のオンラインエディットで変更できる増減サイズに制限はありません。
- ・ 4K ステップを超えるサイズの ST プログラムは、オンラインエディットできません。（CJ2 シリーズを除く）
- ・ 1 回のオンラインエディットで変更できる増減サイズは、0.5K ステップです。（CJ2 シリーズを除く）
- ・ オンラインエディット後、CPU ユニットが内蔵フラッシュメモリへのバックアップを完了するまで（CPU ユニット前面の BKUP LED が点灯から消灯に変わるまで）、PLC 本体の電源を OFF にしないでください。
バックアップ完了前に（CPU ユニット前面の BKUP LED が点灯中に）PLC 本体の電源を OFF にすると、バックアップが行われず、プログラムはオンラインエディット前の状態に戻ります。

付録

付-1 ファンクションブロックで使用するシステム定義の外部変数一覧

種類	フラグ名	CX-Programmer のシステム定義の外部変数名	データ型	アドレス
コンディションフラグ	≧フラグ	P_GE	BOOL	CF000
	≠フラグ	P_NE	BOOL	CF001
	≦フラグ	P_LE	BOOL	CF002
	エラーフラグ	P_ER	BOOL	CF003
	キャリーフラグ	P_CY	BOOL	CF004
	>フラグ	P_GT	BOOL	CF005
	=フラグ	P_EQ	BOOL	CF006
	<フラグ	P_LT	BOOL	CF007
	ネガティブフラグ	P_N	BOOL	CF008
	オーバーフローフラグ	P_OF	BOOL	CF009
	アンダーフローフラグ	P_UF	BOOL	CF010
	アクセスエラーフラグ	P_AER	BOOL	CF011
	常時 OFF フラグ	P_Off	BOOL	CF114
	常時 ON フラグ	P_On	BOOL	CF113
クロックパルス	0.02 秒クロックパルス	P_0_02s	BOOL	CF103
	0.1 秒クロックパルス	P_0_1s	BOOL	CF100
	0.2 秒クロックパルス	P_0_2s	BOOL	CF101
	1 分クロックパルス	P_1min	BOOL	CF104
	1 秒クロックパルス	P_1s	BOOL	CF102
特殊補助リレー	運転開始時 1 サイクル ON フラグ	P_First_Cycle	BOOL	A200.11
	ステップラダー 1 サイクル ON フラグ	P_Step	BOOL	A200.12
	タスク初回起動フラグ	P_First_Cycle_Task	BOOL	A200.15
	サイクルタイム最大値	P_Max_Cycle_Time	UDINT	A262
	サイクルタイム現在値	P_Cycle_Time_Value	UDINT	A264
	サイクルタイムオーバーフラグ	P_Cycle_Time_Error	BOOL	A401.08
	電池異常フラグ	P_Low_Battery	BOOL	A402.04
	I/O 照合異常	P_IO_VerIFy_Error	BOOL	A402.09
	負荷遮断フラグ	P_Output_Off_Bit	BOOL	A500.15
Smart FB ライブラリ専用リレー (*1)	CIO エリア指定	P_CIO	WORD	A450
	HR エリア指定	P_HR	WORD	A452
	WR エリア指定	P_WR	WORD	A451
	DM エリア指定	P_DM	WORD	A460
	EM0～C エリア指定	P_EM0～P_EMC	WORD	A461～A473

*1：Smart FB ライブラリ専用外部変数です。ユーザ作成のファンクションブロックでは使用しないでください。

付-2 ST 言語のエラー一覧

■ エラーメッセージ一覧

エラーメッセージ	エラーメッセージが表示される条件	エラーの例
%s 演算子は%s 型のデータ型ではサポートされていません	演算子でサポートされていないデータ型の変数や数値を使用した。	A:=B+1; (*A、B は WORD 型の変数*)
%s から%s に変換できません	演算結果のデータ型が代入先の変数と一致していない数式や、データ型の異なる変数を代入した。	Y:=ABS(X); (*X は INT 型の変数、Y は UINT 型の変数*)
(がありません	型変換命令や関数の呼び出しにおいて、 "(" がありません。	Y:=INT_TO_DINT X);
) がありません	演算子 () や、型変換命令や関数の呼び出しにおいて、 ")" に対応する "(" がありません。	Y:=(X1+X2/2
0 で除算	0 で割り算をしている数式がある。	
: がありません	CASE 文の整数式の値の後ろに ":" がありません。	CASE A OF 1 X:=1; END_CASE;
:= がありません	代入式において、 ":" がありません。	
; がありません	文の最後に ";" がありません。	
DO がありません	FOR 文、WHILE 文において "DO" がありません。	
END_CASE がありません	CASE 文の最後に "END_CASE" がありません。	
END_FOR がありません	FOR 文の最後に "END_FOR" がありません。	
END_IF がありません	IF 文の最後に "END_IF" がありません。	
END_REPEAT がありません	REPEAT 文の最後に "END_REPEAT" がありません。	
END_WHILE がありません	WHILE 文の最後に "END_WHILE" がありません。	
FOR ループ宣言に無効なパラメータがあります	FOR 文の変数に INT、DINT、LINT、UINT、UDINT、ULINT 以外のデータ型の変数を使用した。	FOR I:=1 TO 100 DO (*I は WORD 型の変数*) X:=X+1; END_FOR;
OF がありません	CASE 文において "OF" がありません。	
THEN がありません	IF 文において "THEN" がありません。	
TO がありません	FOR 文において "TO" がありません。	
UNTIL がありません	REPEAT 文において "UNTIL" がありません。	
[がありません	配列変数の配列インデックスを指定していない。	X:=Array; (*Array は配列変数*)
] がありません	配列インデックスにおいて "[" に対応する "]" がありません。	X:=Array[2]; (*Array は配列変数*)
コメントの最後尾が見つかりません	コメントの始まりを示す "(" に対して、コメントの終わりを示す ")" がありません。	(*コメント
関数の出力変数が不正です。	関数の出力変数の指定が不正 (ENO の転送先として BOOL 型以外の変数または数字を指定した)。	Y:=SIN(X1, ENO=>1);
関数の変数使用不一致	関数のパラメータの使用が不正。	Y:=SIN(X1, EN=>BOOL1); (*入力パラメータ EN を出力パラメータとして使用している*)
関数の変数指定が多過ぎます。	関数のパラメータ指定が多過ぎる。	Y:=SIN(X1, X2);
式が有効ではありません	数式が不正。構文 (IF, WHILE, REPEAT, FOR, CASE) において整数式や条件式が不正あるいは指定されていない場合など。	WHILE DO (*WHILE 文の条件式がない*) X:=X+1; END_WHILE;
数字は NOT 演算ではサポートされていません	数値に対して NOT 演算子を使用した。	Result:=NOT 1;
値が範囲外です	変数にその変数のデータ型で範囲外の値を代入した。	X:=32768; (*X は INT 型の変数*)
定数がありません	CASE 文の整数式において定数が抜けている。	CASE A OF 2..: X:=1; 2..: X:=2; END_CASE;
定数が有効ではありません	CASE 文の整数式に、演算結果が整数型でない数式または整数型でない変数、あるいは数値を指定した。	CASE A OF (*A は REAL 型の変数*) 1: X:=1; 2: X:=2; END_CASE;
入力パラメータがありません。すべての入力パラメータを設定しなければなりません。	関数の引数を指定していない、または指定が少なすぎる。	Y:=EXPT(X);

エラーメッセージ	エラーメッセージが表示される条件	エラーの例
入力変数'%s'は値を割り付けることができません	入力変数に値を代入した。	
配列インデックスが範囲外です	配列サイズより大きい配列インデックスを指定した。	Array[100]:=10; (*Array は配列サイズ 100 の配列変数*)
配列インデックスが有効ではありません	配列インデックスに、演算結果が整数型でない数式または整数型でない変数を指定した。	Array[Index]:=10; (*Index は WORD 型の変数*)
不正な構文 '%s'	キーワード (予約語) や変数が不正に使われている。	FOR I:=1 TO 100 DO BY -1 (*DO の位置が不正*) X:=X+1; END_FOR;
不正な書式'%s'です。	数値の書式が不正。	X:=123.; (*アンダーバーの後ろに数字がない*) X:=1_23; (*アンダーバーが連続して使われている*) X:=2#301; Y:=8#90; (*2,8 進数の数値で使用できない数字を使用している*) 備考: アンダーバーは数字を読みやすくするために数字と数字の間に挿入できる。2#, 8#, 16#を数字の頭につけることで、それぞれ 2,8,16 進数の数字を表現できる。
負の値は%s データ型でサポートされていません	負の値が使えないデータ型 (UINT, UDINT, ULINT) の変数の前にマイナス記号を付けた。	Y:=-X; (*X は UINT 型の変数、Y は INT 型の変数*)
変数 '%s' は読出専用メモリの AT アドレスが設定されているため値を割り付けることができません。	読出専用メモリのアドレス(特殊補助リレーの読出専用エリア、コンディションフラグエリアなど)が割り当てられた変数に値を代入しようとした。	
変数'%s'は関数パラメータではありません。	関数のパラメータに指定できない変数をパラメータに指定した。	Y:=SIN(Z:=X); (*X, Y は REAL 型の変数、Z は SIN 関数のパラメータではない*)
未定義の識別子'%s'	変数テーブルで定義されていない変数を使用した。	
無効な値です	数値が不正。	X:=1e2; (*REAL 型でない数値に対して指数を使用した*) 備考: "e"は 10 の累乗を示す。
命令が有効ではありません	文が不正。構文 (IF, WHILE, REPEAT, FOR, CASE, REPEAT) において、それぞれ "IF", "WHILE", "REPEAT", "FOR", "CASE", "REPEAT"がないなど。	X:=X+1; (*"REPEAT"がない*) UNTIL X>10 END_REPEAT;
有効なコードが最低 1 行必要です(コメントは除く)	コメント以外に一行も有効なコードがない。	

■ ワーニングメッセージ一覧

ワーニングメッセージ	ワーニングメッセージが表示される条件	エラーの例
キーワード'%s'が多過ぎます。	無効な場所でキーワードを使用しているというワーニング。ループ構文の外側で EXIT 文を使用するなど。	
%s'から%s'への変換はデータが失われる可能性があります。	サイズの大きいデータ型からサイズの小さいデータ型に型変換する場合、データが失われる可能性がある。	Y:=DINT_TO_INT(X); (*X は DINT 型の変数、Y は INT 型の変数*)

付-3 関数の説明

付-3-1 標準関数

■ 文字列関数

● LEN: 文字列長の検出

- 機能
指定した文字列の長さを求める。
- 使用方法
戻り値 := LEN(文字列);

・ 引数/戻り値

変数名	データ型	内容
文字列	STRING	長さを求める文字列を指定する。
戻り値	INT	指定された文字列のサイズを返す。

・ 例

[変数]
STRING Message
INT Result

Message

A	B	C	D	E	F	G	H
---	---	---	---	---	---	---	---

Result := LEN(Message);
→ 変数 Result に 8 が格納される。

● LEFT: 左からの文字列抽出

- 機能
指定した文字列に対して左端からの指定文字数分を抽出する。
- 使用方法
戻り値 := LEFT(抽出対象文字列, 抽出文字数);

・ 引数/戻り値

変数名	データ型	内容
抽出対象文字列	STRING	抽出する文字列を指定する。
抽出文字数	INT, UINT	抽出する文字数を指定する。
戻り値	STRING	抽出された文字列を返す。

・ 例

[変数]
STRING Message
STRING Result

Message

A	B	C	D	E	F	G	H
---	---	---	---	---	---	---	---

Result := LEFT(Message, 3);
→ 変数 Result には 'ABC' が格納される。

● RIGHT: 右からの文字列抽出

- 機能
指定した文字列に対して右端からの指定文字数分を抽出する。
- 使用方法
戻り値 := RIGHT(抽出対象文字列, 抽出文字数);

・ 引数/戻り値

変数名	データ型	内容
抽出対象文字列	STRING	抽出する文字列を指定する。
抽出文字数	INT, UINT	抽出する文字数を指定する。
戻り値	STRING	抽出された文字列を返す。

・ 例

[変数]
STRING Message
STRING Result

Message

A	B	C	D	E	F	G	H
---	---	---	---	---	---	---	---

Result := RIGHT(Message, 3);

→ 変数 Result には 'FGH' が格納される。

● MID: 文字列の抽出

・ 機能

指定した文字列に対して指定位置からの指定文字数分を抽出する。

・ 使用方法

戻り値 := MID(抽出対象文字列, 抽出文字数, 抽出位置);

・ 引数/戻り値

変数名	データ型	内容
抽出対象文字列	STRING	抽出する文字列を指定する。
抽出文字数	INT, UINT	抽出する文字数を指定する。
抽出文字位置	INT, UINT	抽出開始位置を指定する。 最初の文字を「1」とする(下図 Message1 の'A'は「1」)。
戻り値	STRING	抽出された文字列を返す。

・ 例

[変数]
STRING Message
STRING Result

Message

A	B	C	D	E	F	G	H
---	---	---	---	---	---	---	---

Result := MID(Message, 3, 2);

→ 変数 Result には 'BCD' が格納される。

● CONCAT: 文字列の結合

・ 機能

指定した文字列を結合する。

指定可能な文字列は最大 31 個。

・ 使用方法

戻り値 := CONCAT(結合文字列 1, 結合文字列 2, ...);

・ 引数/戻り値

変数名	データ型	内容
結合文字列 1	STRING	結合する文字列を指定する。
結合文字列 2	STRING	結合する文字列を指定する。
：		
戻り値	STRING	結合された文字列を返す。

・ 例

[変数]
STRING Message1
STRING Message2
STRING Message3
STRING Result

Message1

A	B	C
---	---	---

Message2

D	E
---	---

Message3

F	G	H
---	---	---

Result := CONCAT(Message1, Message2, Message3);

→ 変数 Result には 'ABCDEFGH' が格納される。

Result

A	B	C	D	E	F	G	H
---	---	---	---	---	---	---	---

● INSERT: 文字列の挿入

- 機能
指定した文字列に対して文字を挿入する。
- 使用方法
戻り値 := INSERT(挿入対象文字列, 挿入文字列, 挿入位置);
- 引数/戻り値

変数名	データ型	内容
挿入対象文字列	STRING	挿入される文字列を指定する。
挿入文字列	STRING	挿入する文字列を指定する。
挿入位置	INT, UINT	挿入する文字の位置を指定する。 最初の文字を「1」とする(下図 Message1 の'A'は「1」)。
戻り値	STRING	挿入された文字列を返す。

・ 例

変数

STRING Message1

STRING Message2

STRING Result

Message1

A

B

C

D

Message2

E

F

G

H

Result := INSERT(Message1, Message2, 2);

→ 変数 Result には 'ABEFGHCD' が格納される。

Result

A

B

E

F

G

H

C

D

● DELETE: 文字列の削除

- 機能
文字列に対して指定位置から指定文字数分を削除する。
- 使用方法
戻り値 := DELETE(削除対象文字列, 削除文字数, 削除位置);
- 引数/戻り値

変数名	データ型	内容
削除対象文字列	STRING	削除される文字列を指定する。
削除文字数	INT, UINT	削除する文字数を指定する。
削除位置	INT, UINT	削除する文字の位置を指定する。 最初の文字を「1」とする(下図 Message1 の'A'は「1」)。
戻り値	STRING	削除された文字列を返す。

・ 例

変数

STRING Message1

STRING Result

Message

A

B

C

D

E

F

G

H

Result := DEL(Message1, 4, 2);

→ 変数 Result には'AFGH'が格納される。

Result

A

F

G

H

● REPLACE: 文字列の置換

- 機能
文字列に対して指定位置から指定文字数分の文字を変更する。
- 使用方法
戻り値 := REPLACE(置換対象文字列, 置換文字列, 置換文字数, 置換位置);

・ 引数/戻り値

変数名	データ型	内容
置換対象文字列	STRING	置換される文字列を指定する。
置換文字列	STRING	置換する文字列を指定する。
置換文字数	INT, UINT	置換される文字数を指定する。
置換位置	INT, UINT	置換される文字の位置を指定する。 最初の文字を「1」とする(下図 Message1 の'A'は「1」)。
戻り値	STRING	置換された文字列を返す。

・ 例

[変数]

STRING Message1
STRING Message2
STRING Result

Message1

A	B	C	D	E	F	G	H
---	---	---	---	---	---	---	---

Message2

X	Y	Z
---	---	---

Result := REPLACE(Message1, Message2, 2, 3);
→ 変数 Result には 'ABXYEFGH' が格納される。

Result

A	B	X	Y	E	F	G	H
---	---	---	---	---	---	---	---

● FIND: 文字列の検索

・ 機能

指定された文字列に対して検索文字列を検索し、最初に発見した文字位置を返す。

合致しない場合はゼロを返す。

・ 使用方法

戻り値 := FIND(検索対象文字列, 検索文字列);

・ 引数/戻り値

変数名	データ型	内容
検索対象文字列	STRING	検索される文字列を指定する。
検索文字列	STRING	検索する文字列を指定する。
戻り値	INT	最初に発見した文字の位置を返す。 最初の文字を「1」とする(下図 Message1 の'A'は「1」)。

・ 例

[変数]

STRING Message1
STRING Message2
INT Result

Message1

A	B	C	D	E	C	D	E
---	---	---	---	---	---	---	---

Message2

C	D	E
---	---	---

Result := FIND(Message1, Message2);
→ 変数 Result には 3 が格納される。

■ データシフト関数

● SHL: 左ビットシフト

- 機能
ビット文字列を左へ n ビットシフトします。
シフトしたとき、右にはゼロが入ります。
- 使用方法
戻り値 := SHL (シフト対象データ, ビット数);

・ 引数/戻り値

変数名	データ型	内容
シフト対象データ S1	BOOL, WORD, DWORD, LWORD	シフトされるデータ
ビット数 n	INT, UINT, UDINT, ULINT, DINT, LINT,	シフトするビット数
戻り値	BOOL, WORD, DWORD, LWORD	出力データ

- 備考
第 1 引数, 戻り値は同じデータ型を設定する必要があります。
- 例

[変数]
WORD Data1 Data1 = B26E(16 進数)→1011 0010 0110 1110(2 進数)
INT N N = 1(10 進数)
WORD Result

Data1	1	0	1	1	0	0	1	0	0	1	1	0	1	1	1	0
Result	0	1	1	0	0	1	0	0	1	1	0	1	1	1	0	0

Result := SHL(Data1, N);
→ 変数 Result には'64DC' が格納される。

● SHR: 右ビットシフト

- 機能
ビット文字列を右へ n ビットシフトします。
シフトしたとき、左にはゼロが入ります。
- 使用方法
戻り値 := SHR (シフト対象データ S1, ビット数 n);

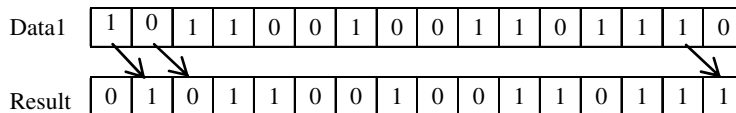
・ 引数/戻り値

変数名	データ型	内容
シフト対象データ S1	BOOL, WORD, DWORD, LWORD	シフトされるデータ
ビット数 n	INT, UINT, UDINT, ULINT, DINT, LINT,	シフトするビット数
戻り値	BOOL, WORD, DWORD, LWORD	出力データ

- 備考
第 1 引数, 戻り値は同じデータ型を設定する必要があります。

・ 例

[変数]
 WORD Data1 Data1 = B26E(16 進数)→1011 0010 0110 1110(2 進数)
 INT N N = 1(10 進数)
 WORD Result



Result := SHR(Data1,N);
 → 変数Result に'5937'が格納される。

● ROL: 左ビット回転

・ 機能

ビット文字列を左へ n ビット回転します。

・ 使用方法

戻り値 := ROL (回転対象データ S1, ビット数 n);

・ 引数/戻り値

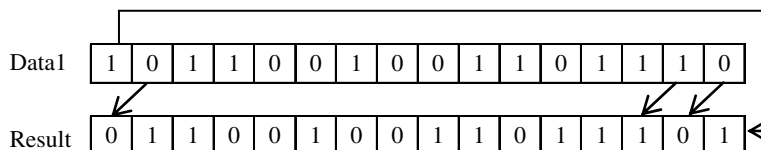
変数名	データ型	内容
回転対象データ S1	BOOL, WORD, DWORD, LWORD	回転されるデータ
ビット数 n	INT, UINT, UDINT, ULINT, DINT, LINT,	回転するビット数
戻り値	BOOL, WORD, DWORD, LWORD	出力データ

・ 備考

第 1 引数, 戻り値は同じデータ型を設定する必要があります。

・ 例

[変数]
 WORD Data1 Data1 = B26E(16 進数)→1011 0010 0110 1110(2 進数)
 INT N N = 1(10 進数)
 WORD Result



Result := ROL(Data1,N);
 → 変数Result に'64DD'が格納される。

● ROR: 右ビット回転

・ 機能

ビット文字列を右へ n ビット回転します。

・ 使用方法

戻り値 := ROR (回転対象データ S1, ビット数 n);

・ 引数/戻り値

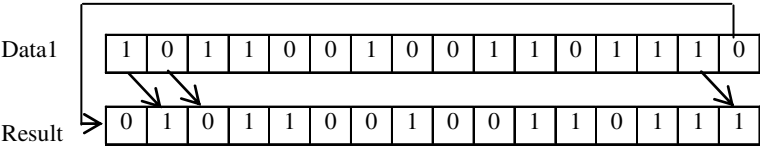
変数名	データ型	内容
回転対象データ S1	BOOL, WORD, DWORD, DWORD	回転されるデータ
ビット数 n	INT, UINT, UDINT, ULINT, DINT, LINT,	回転するビット数
戻り値	BOOL, WORD, DWORD, DWORD	出力データ

・ 備考

第 1 引数, 戻り値は同じデータ型を設定する必要があります。

・ 例

[変数]
WORD Data1 Data1 = B26E(16 進数)→1011 0010 0110 1110(2 進数)
INT N N = 1(10 進数)
WORD Result



Result := ROR(Data1,N);
→ 変数 Result には'5937' が格納される。

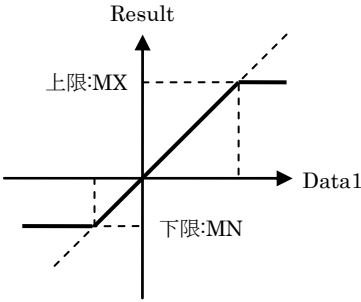
■ データ制御関数

● LIMIT: 上下限リミット制御

- 機能
入力データが上下限リミットデータの範囲内かどうかにより、出力データを制御します。
- 使用方法
戻り値 := LIMIT (下限リミットデータ, 入力データ, 上限リミットデータ);
- 引数/戻り値

変数名	データ型	内容
下限リミットデータ	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	下限リミット
入力データ	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	入力データ
上限リミットデータ	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	上限リミット
戻り値	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	出力データ

- 備考
入力データ < 下限リミットデータの時、戻り値は下限リミットデータをとります。
上限リミットデータ < 入力データの時、戻り値は上限リミットデータをとります。
下限リミットデータ ≤ 入力データ ≤ 上限リミットデータ
の時、戻り値は入力データをとります。
引数、戻り値は同じデータ型を設定する必要があります。



- 例
[変数] MN = 123 Data1 = 456 MX = 789
 INT MN
 INT Data1
 INT MX Result := LIMIT (MN, Data1, MX);
 INT Result → 変数 Result には '456' が格納される。

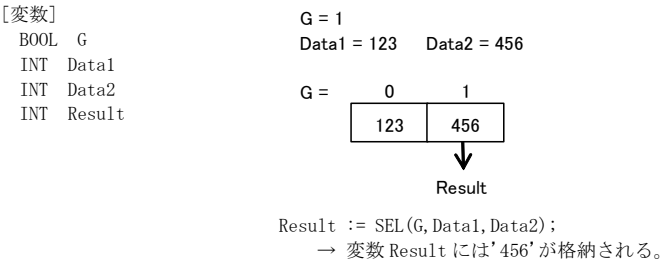
■ データ選択関数

● SEL: データの選択

- 機能
2つのデータから選択条件で指定された1つのデータを選択します。
- 使用方法
戻り値 := SEL (選択条件, 選択対象データ S1, 選択対象データ S2);
- 引数/戻り値

変数名	データ型	内容
選択条件 g	BOOL	選択条件 g が真(=1)ならば S2 を選択, 偽(=0)ならば S1 を選択する
選択対象データ S1	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	選択対象データ
選択対象データ S2	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	選択対象データ
戻り値	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	出力データ

- 備考
選択対象データは同じデータ型を設定する必要があります。
- 例



● MUX: マルチプレクサ

- 機能
抽出条件から指定されたデータを抽出します。
指定できるデータ数は最大 30 個。
- 使用方法
戻り値 := MUX (抽出条件, 抽出対象データ S1, 抽出対象データ S2, ...);

・ 引数/戻り値

変数名	データ型	内容
抽出条件 n	INT, UINT, UDINT, ULINT, DINT, LINT	抽出するデータを指定する。 選択条件 n=0,1,2,...,29 n=0 の時、抽出対象データ=S1 n=1 の時、抽出対象データ=S2 : n=n の時、抽出対象データ=S(n+1)
抽出対象データ S1	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	抽出対象データ
抽出対象データ S2	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	抽出対象データ
:		
戻り値	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	出力データ

・ 備考

戻り値 := MUX(選択条件, 抽出対象データ S1, 抽出対象データ S2, ...);

抽出条件である第 1 引数を除き、全ての引数と戻り値は同じデータ型を設定する必要があります。

抽出条件が 0～29 以外の時、不定値が戻り値に格納されます。

・ 例

[変数]
INT N
INT Data1
INT Data2
INT Data3
INT Result

N = 0
Data1 = 123 Data2 = 456 Data3 = 789

N = 0 1 2
 ┌───┬───┬───┐
 │ 123 │ 456 │ 789 │
 └───┴───┴───┘

↓
Result

Result := MUX(N, Data1, Data2, Data3);
→ 変数 Result には '123' が格納される。

● MAX: 最大値

・ 機能

対象データから最大値を選択します。

指定できるデータ数は最大 31 個です。

・ 使用方法

戻り値 := MAX (対象データ S1, 対象データ S2, 対象データ S3, ..., 対象データ S31);

・ 引数/戻り値

変数名	データ型	内容
対象データ S1	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	対象データ
対象データ S2	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	対象データ
抽出対象データ S3	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	対象データ
⋮		
対象データ S31	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	対象データ
戻り値	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	出力データ

・ 備考

戻り値 := MAX(対象データ S1, 対象データ S2, 対象データ S3, ..., 対象データ S31);
全ての引数と戻り値は同じデータ型を設定する必要があります。

・ 例

[変数]
INT Data1
INT Data2
INT Data3
INT Result

Data1 = 123 Data2 = 456 Data3 = 789

123	456	789
-----	-----	-----

↓
Result

Result := MAX(Data1, Data2, Data3);
→ 変数 Result には'789'が格納される。

● MIN: 最小値

・ 機能

対象データから最小値を選択します。
指定できるデータ数は最大 31 個です。

・ 使用方法

戻り値 := MIN (対象データ S1, 対象データ S2, 対象データ S3, ..., 対象データ S31);

・ 引数/戻り値

変数名	データ型	内容
対象データ S1	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	対象データ
対象データ S2	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	対象データ
対象データ S3	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	対象データ
:		
対象データ S31	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	対象データ
戻り値	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	出力データ

・ 備考

戻り値 := MIN(対象データ S1, 対象データ S2, 対象データ S3, ..., 対象データ S31);
 全ての引数と戻り値は同じデータ型を設定する必要があります。

・ 例

[変数]

```
INT Data1
INT Data2
INT Data3
INT Result
```

Data1 = 123 Data2 = 456 Data3 = 789

123	456	789
-----	-----	-----



Result

```
Result := MIN(Data1, Data2, Data3);
→ 変数 Result には'123'が格納される。
```

付-3-2 オムロン拡張関数

■ メモリカード処理関数

● WRITE_TEXT: テキストファイルの作成

- 機能
指定した文字列をメモリカードの指定ファイルに書き込む。
- 使用方法
WRITE_TEXT(書出文字列, ディレクトリ名+ファイル名, 区切り文字, パラメータ);
- 引数/戻り値

変数名	データ型	内容
書出文字列	STRING	
ディレクトリ名 +ファイル名	STRING	ルートディレクトリ(¥)を含めて指定します。 ファイル名は最大8文字です。 拡張子はTXT固定となります。 例えば、ルートディレクトリにファイル名 LINE_A.TXTを作成する場合、¥LINE_A
区切り文字	STRING	':': 空文字 ';': カンマ '\$L'または'\$!': 改行(ASCIIコード: #0A) '\$N'または'\$n': 復帰改行(ASCIIコード: #0D0A) '\$P'または'\$p': 改ページ(ASCIIコード: #0C) '\$R'または'\$r': 復帰(ASCIIコード: #0D) '\$T'または'\$t': タブ(ASCIIコード: #09)
パラメータ	INT, UINT, WORD	0: 追加 1: 新規作成

・ 例

[変数]		
BOOL	P_MemCardBusyFlag	(*メモリカード操作実行中フラグ*) AT A343.13
BOOL	P_MemCardAckFlag	(*メモリカード認識完了フラグ*) AT A343.15
STRING	FileName	(*ファイル名*)
INT	LogData1 2 3	(*ログ数値*)
STRING	FieldStr1 2 3	(*ログ数値文字列*)
STRING	CsvLineStr	(*CSV形式ログ1行文字列*)

```
FileName := '¥LOGFILE';
LogData1 := 12;
LogData2 := 345;
LogData3 := 6789;

(*メモリカードの書き込み条件が成立したら、テキストファイルに出力する*)
IF ( P_MemCardAckFlag AND (NOT P_MemCardBusyFlag) ) THEN

    (*数値→文字列の変換*)
    FieldStr1 := INT_TO_STRING( LogData1 );
    FieldStr2 := INT_TO_STRING( LogData2 );
    FieldStr3 := INT_TO_STRING( LogData3 );
    (*数値データ1行文字列[CSV形式]の作成*)
    CsvLineStr := FieldStr1 + ',' + FieldStr2 + ',' + FieldStr3;
    (*数値データ1行をファイル出力*)
    WRITE_TEXT( CsvLineStr, FileName, '$n', 0 );
END_IF;
```

出力されるファイルの内容

LOGFILE.TXT
12,345,6789

付-3 関数の説明
付-3-2 オムロン拡張関数

関連特殊補助リレー	アドレス	
ファイルメモリ操作実行中フラグ	A343.13	以下のいずれかのときに TRUE となります。 ・ 自 CPU ユニット宛 CMND 命令実行中 ・ ファイルメモリ用命令実行中 ・ 特殊補助リレーによるプログラム差し替え中 ・ 簡易バックアップ操作中
メモリカード認識完了フラグ	A343.15	メモリカードの認識ができると TRUE となります。

その他の関連特殊補助リレーに関する情報や留意事項は「CS/CJ シリーズ コマンドリファレンス」のファイル用命令 TWRIT の項をご参照ください。

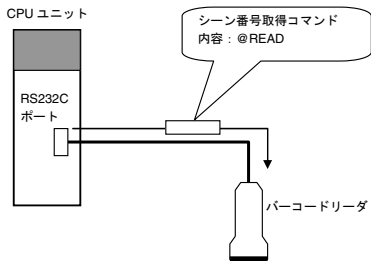
■ 通信処理関数

● TXD_CPU: 文字列の送信（CPU ユニットの RS-232C ポート）

- 機能
CPU ユニットの RS-232C ポートから文字列を送信する。
- 使用方法
TXD_CPU（送信文字列）；
- 条件
RS-232C ポートのシリアル通信モードを「無手順」に設定します。
- 引数/戻り値

変数名	データ型	内容
送信文字列	STRING	送信する文字列

・ 例



[変数]

BOOL	DoSendData	(* 送信指令 *)
INT	iProcess	(* 処理番号 *)
STRING	Message	(* 送信メッセージ *)
BOOL	SendEnableCPUPort	(* 送信レディフラグ *) AT A392.05

```
(* DoSendDataフラグがONかつiProcessが0のときデータを送信する *)
IF (DoSendData = TRUE) AND (iProcess = 0) THEN
  iProcess := 1;
  DoSendData := FALSE;
END_IF;
(* 処理番号に従って送信処理を行う *)
CASE iProcess OF
  1: (* 送信用文字データの作成 *)
    Message := '@READ';
    iProcess := 2;
  2: (* 送信可能ならば送信関数を実行 *)
    IF SendEnableCPUPort = TRUE THEN
      TXD_CPU(Message);
      iProcess := 3;
    END_IF;
  3: (* 送信レディフラグがONすると送信完了 *)
    IF SendEnableCPUPort = TRUE THEN
      iProcess := 0;
    END_IF;
END_CASE;
```

関連特殊補助リレー	アドレス	
RS-232C ポート送信レディフラグ	A392.05	無手順モードで、送信可能なとき TRUE となります。

その他の関連特殊補助リレーに関する情報や留意事項は「CS/CJ シリーズ コマンドリファレンス」のシリアル通信命令 TXD の項をご参照ください。

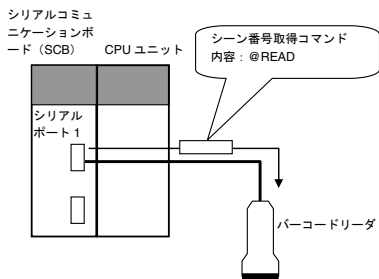
● TXD_SCB: 文字列の送信（シリアルコミュニケーションボード（SCB）のシリアルポート）

- 機能
シリアルコミュニケーションボード（SCB）のシリアルポートから文字列を送信する。
- 使用方法
TXD_SCB（送信文字列, シリアルポート番号）；
- 条件
シリアルポートのシリアル通信モードを「無手順」に設定します。

・ 引数/戻り値

変数名	データ型	内容
送信文字列	STRING	送信する文字列
シリアルポート番号	INT, UINT, WORD	シリアルポート番号 1: シリアルポート 1 2: シリアルポート 2

・ 例



[変数]
BOOL P_DoSendData (* 送信指令 *)
INT iProcess (* 処理番号 *)
STRING Message (* 送信メッセージ *)
BOOL P_SendEnableSCBPort1 (* 送信レディフラグ *) AT A356.05 シリアルポート番号 1 を使用

```
(* シリアルポート番号: 1 を使用 *)  
(* P_DoSendData フラグが ON かつ iProcess が 0 のときデータを送信する *)  
IF (P_DoSendData = TRUE) AND (iProcess = 0) THEN  
  iProcess := 1;  
  P_DoSendData := FALSE;  
END_IF;  
(* 処理番号に従って送信処理を行う *)  
CASE iProcess OF  
  1: (* 送信用文字データの作成 *)  
    Message := '@READ';  
    iProcess := 2;  
  2: (* 送信可能ならば送信関数を実行 *)  
    IF P_SendEnableSCBPort1 = TRUE THEN  
      TXD_SCB(Message, 1);  
      iProcess := 3;  
    END_IF;  
  3: (* 送信レディフラグが ON すると送信完了 *)  
    IF P_SendEnableSCBPort1 = TRUE THEN  
      iProcess := 0;  
    END_IF;  
END_CASE;
```

関連特殊補助リレー	アドレス	
ポート 1 送信レディフラグ	A356.05	無手順モードで、送信可能なとき TRUE となります。
ポート 2 送信レディフラグ	A356.13	無手順モードで、送信可能なとき TRUE となります。

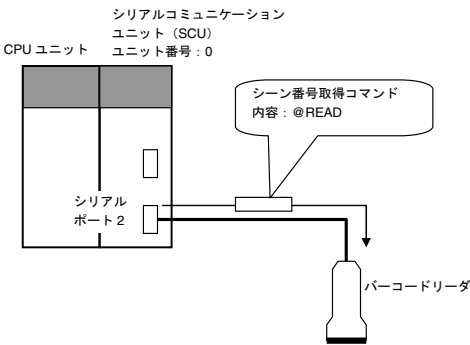
その他の関連特殊補助リレーに関する情報や留意事項は「CS/CJ シリーズ コマンドリファレンス」のシリアル通信命令 TXD の項をご参照ください。

● TXD_SCU: 文字列の送信（シリアルコミュニケーションユニット（SCU）シリアルポート

- 機能
シリアルコミュニケーションユニット（SCU）のシリアルポートから文字列を送信する。
- 使用方法
TXD_SCU（送信文字列, シリアルコミュニケーションユニットのユニット番号, シリアルポート番号, 内部論理ポート番号）；
- 条件
シリアルポートのシリアル通信モードを「無手順」に設定します。
- 引数/戻り値

変数名	データ型	内容
送信文字列	STRING	送信する文字列
シリアルコミュニケーションユニットのユニット番号	INT, UINT, WORD	
シリアルポート番号	INT, UINT, WORD	1: シリアルポート 1 2: シリアルポート 2
内部論理ポート番号	INT, UINT, WORD	0 ～ 7: 内部論理ポート番号指定時 16#F: 内部論理ポート自動割付時

例



[変数]
 BOOL P_DoSendData (* 送信指令 *)
 INT iProcess (* 工程番号 *)
 STRING Message (* 送信メッセージ *)
 BOOL P_TXDU_Exe (* TXDU 命令実行中フラグ *) AT 1519.05 ユニット番号 0、シリアルポート番号 2 を使用
 BOOL P_ComInstEnable (* ネットワーク通信命令実行可フラグ *) AT A202.07 論理ポート番号 7 を使用

(* ユニット番号: 0 シリアルポート番号: 2 論理ポート番号: 7 を使用 *)
 (* P_DoSendDataフラグがONかつiProcessが0のときデータを送信する *)
 IF (P_DoSendData = TRUE) AND (iProcess = 0) THEN
 iProcess := 1;
 P_DoSendData := FALSE;
 END_IF;

 (* 処理番号に従って送信処理を行う *)
 CASE iProcess OF
 1: (* 送信用文字データの作成 *)
 Message := '@READ';
 iProcess := 2;
 2: (* 通信可フラグがONかつTXDU命令非実行中ならば送信関数を実行 *)
 IF (P_ComInstEnable = TRUE) AND (P_TXDU_Exe = FALSE) THEN
 TXD_SCU(Message, 0, 2, 7);
 iProcess := 3;
 END_IF;
 3: (* 通信可フラグがONすると送信完了 *)
 IF P_ComInstEnable = TRUE THEN
 iProcess := 0;
 END_IF;
 END_CASE;

関連特殊補助リレー	アドレス	
ネットワーク通信命令実行可フラグ	A202.00～ A202.07	ネットワーク通信が実行可能であるとき TRUE となります。 各ビットが内部論理ポート番号に対応します。 ビット 00～07: 内部論理ポート 0～7

関連 CPU 高性能ユニットリレーエリア	ビット	
n=1500+25*ユニット番号 ポート 1: n+9 ポート 2: n+19	05	TXDU 命令が実行中のとき TRUE となります。

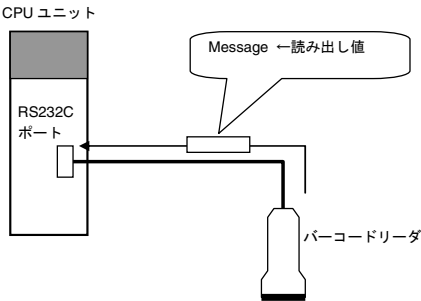
その他の関連特殊補助リレーに関する情報や留意事項は「CS/CJ シリーズ コマンドリファレンス」のシリアル通信命令 TXDU の項をご参照ください。

● RXD_CPU: 文字列の受信 (CPU ユニットの RS-232C ポート)

- 機能
CPU ユニットの RS-232C ポートから受信した文字列を格納する。
- 使用方法
RXD_CPU (受信文字列格納先, 受信文字数) ;
- 条件
RS-232C ポートのシリアル通信モードを「無手順」に設定します。
- 引数/戻り値

変数名	データ型	内容
受信文字列格納先	STRING	受信文字列の格納先を指定する
受信文字数	INT, UINT, WORD	受信文字数を指定する 0~255

例



[変数]
BOOL P_DoRecvData (* 受信指令 *)
STRING Message (* 受信メッセージ格納先 *)
BOOL P_EndRecvCPUPort (* 受信完了フラグ *) AT A392.06

(* P_DoRecvData フラグが ON かつ受信が完了しているときデータを受信する *)
IF (P_DoRecvData = TRUE) AND (P_EndRecvCPUPort = TRUE) THEN
(* 16 文字取得 *)
RXD_CPU(Message, 16);
P_DoRecvData := FALSE;
END_IF;

関連特殊補助リレー	アドレス	
RS-232C ポート 受信完了レディフラグ	A392.06	無手順モードで、受信を完了したとき TRUE となります。
RS-232C ポート 受信オーバーフラグ	A392.07	無手順モードで、受信データ数をオーバーして受信したとき TRUE となります。
RS-232C ポート 受信カウンタ	A393	無手順モードで、受信した文字数を表示します。

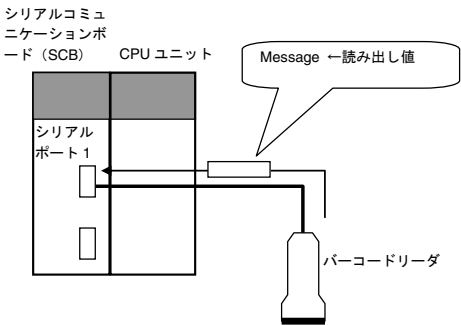
その他の関連特殊補助リレーに関する情報や留意事項は「CS/CJ シリーズ コマンドリファレンス」のシリアル通信命令 RXD の項をご参照ください。

● RXD_SCB: 文字列の受信（シリアルコミュニケーションボード（SCB）のシリアルポート）

- 機能
シリアルコミュニケーションボード（SCB）のシリアルポートから受信した文字列を格納する。
- 使用方法
RXD_SCB（受信文字列格納先, 受信文字数, シリアルポート番号）；
- 条件
シリアルポートのシリアル通信モードを「無手順」に設定します。
- 引数/戻り値

変数名	データ型	内容
受信文字列格納先	STRING	受信文字列の格納先を指定する
受信文字数	INT, UINT, WORD	受信文字数を指定する 0~255
シリアルポート番号	INT, UINT, WORD	シリアルポート番号 1: シリアルポート 1 2: シリアルポート 2

・ 例



[変数]

BOOL P_DoRecvData (* 受信指令 *)
STRING Message (* 受信メッセージ格納先 *)
BOOL P_EndRecvSCBPort1 (* 受信完了フラグ *) AT A356.06 シリアルポート番号 1 を使用

(* シリアルポート番号 : 1 を使用 *)
(* P_DoRecvData フラグが ON かつ受信が完了しているときデータを受信する *)
IF (P_DoRecvData = TRUE) AND (P_EndRecvSCBPort1 = TRUE) THEN
(* 16文字取得 *)
RXD_SCB(Message, 16, 1);
P_DoRecvData := FALSE;
END_IF;

関連特殊補助リレー	アドレス	
ポート 1 受信完了レディフラグ	A356.06	無手順モードで、受信を完了したとき TRUE となります。
ポート 1 受信オーバーフラグ	A356.07	無手順モードで、受信データ数をオーバーして受信したとき TRUE となります。
ポート 1 受信カウンタ	A357	無手順モードで、受信した文字数を表示します。
ポート 2 受信完了レディフラグ	A356.14	無手順モードで、受信を完了したとき TRUE となります。
ポート 2 受信オーバーフラグ	A356.15	無手順モードで、受信データ数をオーバーして受信したとき TRUE となります。
ポート 2 受信カウンタ	A358	無手順モードで、受信した文字数を表示します。

その他の関連特殊補助リレーに関する情報や留意事項は「CS/CJ シリーズ コマンドリファレンス」のシリアル通信命令 RXD の項をご参照ください。

● RXD_SCU: 文字列の受信（シリアルコミュニケーションユニット（SCU）のシリアルポート）

・機能

シリアルコミュニケーションユニット（SCU）のシリアルポートから受信した文字列を格納する。

・使用方法

RXD_SCU（受信文字列格納先, 受信文字数, シリアルコミュニケーションユニットのユニット番号, シリアルポート番号, 内部論理ポート番号）；

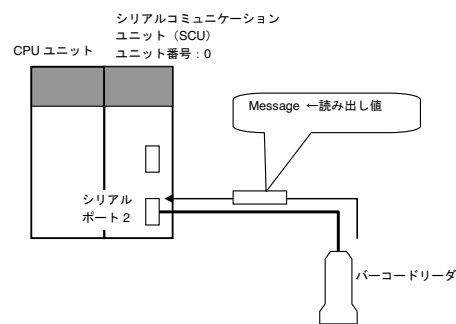
・条件

シリアルポートのシリアル通信モードを「無手順」に設定します。

・引数/戻り値

変数名	データ型	内容
受信文字列格納先	STRING	受信文字列の格納先を指定する
受信文字数	INT, UINT, WORD	受信文字数を指定する 0～255
シリアルコミュニケーションユニットのユニット番号	INT, UINT, WORD	
シリアルポート番号	INT, UINT, WORD	1: シリアルポート 1 2: シリアルポート 2
内部論理ポート番号	INT, UINT, WORD	0 ～ 7: 内部論理ポート番号指定時 16#F: 内部論理ポート自動割付時

・例



[変数]
 BOOL P_DoRecvData (* 受信指令 *)
 INT iProcess (* 工程番号 *)
 STRING Message (* 受信メッセージ *)
 BOOL P_RXDU_Recv (* 受信完了フラグ *) AT 1519.06 ユニット番号 0、シリアルポート番号 2 を使用
 BOOL P_ComInstEnable (* ネットワーク通信命令実行可フラグ *) AT A202.07 論理ポート番号 7 を使用

(* ユニット番号: 0 シリアルポート番号: 2 論理ポート番号: 7 を使用 *)
 (* P_DoRecvData フラグが ON かつ iProcess が 0 のときデータを受信する *)
 IF (P_DoRecvData = TRUE) AND (iProcess = 0) THEN
 iProcess := 1;
 P_DoRecvData := FALSE;
 END_IF;

 (* 工程番号に従って受信処理を行う *)
 CASE iProcess OF
 1: (* 通信可フラグが ON かつ受信完了フラグが ON ならば受信関数を実行 *)
 IF (P_ComInstEnable = TRUE) AND (P_RXDU_Recv = TRUE) THEN
 RXD_SCU(Message, 16, 0, 2, 7);
 iProcess := 2;
 END_IF;
 2: (* 通信可フラグが ON すると受信完了 *)
 IF P_ComInstEnable = TRUE THEN
 iProcess := 0;
 END_IF;
 END_CASE;

関連特殊補助リレー	アドレス	
ネットワーク通信命令実行可フラグ	A202.00～ A202.07	ネットワーク通信が実行安濃であるとき TRUE となります。 各ビットが内部論理ポート番号に対応します。 ビット 00～07: 内部論理ポート 0～7

関連 CPU 高機能ユニットリレーエリア	ビット	
n=1500+25*ユニット番号 ポート 1: n+9 ポート 2: n+19	06	無手順モードで、受信を完了したとき TRUE となります。
n=1500+25*ユニット番号 ポート 1: n+9 ポート 2: n+19	07	無手順モードで、受信データ数をオーバーして受信したとき TRUE となります。
n=1500+25*ユニット番号 ポート 1: n+10 ポート 2: n+20	—	無手順モードで、受信した文字数を表示します。

その他の関連特殊補助リレーに関する情報や留意事項は「CS/CJ シリーズ コマンドリファレンス」のシリアル通信命令 RXDU の項をご参照ください。

■ 角度変換関数

● DEG_TO_RAD: 角度→ラジアン変換

- 機能
角度 (°) をラジアンに変換する。
- 使用方法
戻り値:= DEG_TO_RAD(引数);
- 引数/戻り値

変数名	データ型	内容
引数	REAL LREAL	角度 (°)
戻り値	REAL LREAL	ラジアン

● RAD_TO_DEG: ラジアン→角度変換

- 機能
ラジアンを角度 (°) に変換する。
- 使用方法
戻り値:= RAD_TO_DEG(引数);
- 引数/戻り値

変数名	データ型	内容
引数	REAL LREAL	ラジアン
戻り値	REAL LREAL	角度 (°)

■ タイマ/カウンタ関数

● TIMX: 100ms タイマ

・ 機能

減算式 ON デイレー100ms 単位のタイマの動作をします。

(ラダー命令 TIMX (550) と同様)

- ・ 実行条件が FALSE から TURE になった時点で、タイマアドレスで指定したタイマを起動し、タイマ設定値で指定した値を元に現在値を 100ms おきに減算(-1)します。
- ・ 実行条件が TURE の間、タイマ現在値の更新を実施し、現在値が 0 になると指定したタイマアドレスのタイムアップフラグが ON します。現在値が 0 でなければ、タイムアップフラグが OFF となります。
- ・ 実行条件が FALSE のときは、タイマアドレスの現在値に設定値をセットし、タイムアップフラグを OFF にします。

・ 使用方法

TIMX(実行条件、タイマアドレス、タイマ設定値);

・ 引数

変数名	データ型	内容
実行条件	BOOL	実行条件が ON の間タイマ動作が実行されます。
タイマアドレス	TIMER	使用するタイマアドレス(T0～T4095)変数を指定します。
タイマ設定値	UINT	デイレー時間を 100ms 単位で指定します。 (&0~&65535、#0~#FFFF)

・ 備考

- ・ PLC のプロパティの設定「T2048-4095 の TIM 実行を他の TIM と同じにする」が ON の場合のみ全サイクル実行タスク実行終了時および 80ms ごとに現在値の更新を実施します。
- ・ タイムアップフラグをユーザプログラムで参照する場合には、タイミングによっては 1 サイクルタイム分の遅れが生じることがあります。

● TIMHX: 10ms タイマ (高速タイマ)

・ 機能

減算式 ON デイレー10ms 単位のタイマの動作をします。

(ラダー命令 TIMHX (551) と同様)

- ・ 実行条件が FALSE から TURE になった時点で、タイマアドレスで指定したタイマを起動し、タイマ設定値で指定した値を元に現在値を 10ms おきに減算(-1)します。
- ・ 実行条件が TURE の間、タイマ現在値の更新を実施し、現在値が 0 になると指定したタイマアドレスのタイムアップフラグが ON します。現在値が 0 でなければ、タイムアップフラグが OFF となります。
- ・ 実行条件が FALSE のときは、タイマアドレスの現在値に設定値をセットし、タイムアップフラグを OFF にします。

・ 使用方法

TIMHX(実行条件、タイマアドレス、タイマ設定値);

・ 引数

変数名	データ型	内容
実行条件	BOOL	実行条件が ON の間タイマ動作が実行されます。
タイマアドレス	TIMER	使用するタイマアドレス(T0～T4095)変数を指定します。
タイマ設定値	UINT	デイレー時間を 10ms 単位で指定します。 (&0~&65535、#0~#FFFF)

- ・ 備考
 - ・ PLC のプロパティの設定「T2048-4095 の TIM 実行を他の TIM と同じにする」が ON の場合のみ全サイクル実行タスク実行終了時または 80ms ごとに現在値の更新を実施します。
 - ・ タイマ現在値をユーザプログラムで参照する場合、タイマ番号 0～255、256～2047、2048～4095の間では、更新タイミングの違いから得られる現在値に違いが発生することがあります。
 - ・ タイムアップフラグをユーザプログラムで参照する場合には、タイミングによっては 1 サイクルタイム分の遅れが生じることがあります。

● TMHHX: 1ms タイマ（超高速タイマ）

- ・ 機能

減算式 ON デイレー1ms 単位のタイマの動作をします。
(ラダー命令 TMHHX (552) と同様)

 - ・ 実行条件が FALSE から TURE になった時点で、タイマアドレスで指定したタイマを起動し、タイマ設定値で指定した値を元に現在値を 1ms おきに減算(-1)します。
 - ・ 実行条件が TURE の間、タイマ現在値の更新を実施し、現在値が 0 になると指定したタイマアドレスのタイムアップフラグが ON します。現在値が 0 でなければ、タイムアップフラグが OFF となります。
 - ・ 実行条件が FALSE のときは、タイマアドレスの現在値に設定値をセットし、タイムアップフラグを OFF にします。
- ・ 使用方法

TMHHX(実行条件、タイマアドレス、タイマ設定値);

・ 引数

変数名	データ型	内容
実行条件	BOOL	実行条件が ON の間タイマ動作が実行されます。
タイマアドレス	TIMER	使用するタイマアドレス(T0～T4095)変数を指定します。
タイマ設定値	UINT	デイレー時間を 1ms 単位で指定します。 (&0~&65535、#0~#FFFF)

- ・ 備考
 - ・ PLC のプロパティの設定「T2048-4095 の TIM 実行を他の TIM と同じにする」が ON の場合のみ全サイクル実行タスク実行終了時に現在値の更新を実施します。
 - ・ タイマ現在値をユーザプログラムで参照する場合、タイマ番号 16 以降の場合には、更新が命令実行時のみのため、1ms ごとに更新が実行されるタイマ番号 0～15 と比較して、得られる現在値に違いが発生することがあります。
 - ・ タイムアップフラグをユーザプログラムで参照する場合には、タイミングによっては 1 サイクルタイム分の遅れが生じることがあります。

● TIMUX: 0.1ms タイマ

・機能

減算式 ON ディレー0.1ms 単位のタイマの動作をします。

(ラダー命令 TIMUX (556) と同様)

- ・実行条件が FALSE から TRUE になった時点で、タイマアドレスで指定したタイマを起動し、タイマ設定値で指定した値を元に現在値を 0.1ms おきに減算(-1)します。
- ・実行条件が TRUE の間、タイマ現在値の更新を実施し、現在値が 0 になると指定したタイマアドレスのタイムアップフラグが ON します。現在値が 0 でなければ、タイムアップフラグが OFF となります。
- ・実行条件が FALSE のときは、タイマアドレスの現在値に設定値をセットし、タイムアップフラグを OFF にします。

・使用方法

TIMUX(実行条件、タイマアドレス、タイマ設定値);

・引数

変数名	データ型	内容
実行条件	BOOL	実行条件が ON の間タイマ動作が実行されます。
タイマアドレス	TIMER	使用するタイマアドレス(T0～T4095)変数を指定します。
タイマ設定値	UINT	ディレー時間を 0.1ms 単位で指定します。 (&0～&65535、#0～#FFFF)

・備考

- ・サイクルタイムが 100ms 以上のとき正常に動作しないことがあります。
- ・タイマ現在値をユーザプログラムで参照する場合には、タイミングによっては、実値と 1 サイクルタイム分のずれが生じることがあります。
- ・タイムアップフラグをユーザプログラムで参照する場合には、タイミングによっては、1 サイクルタイム分の遅れが生じることがあります。

● TMUHX: 0.01ms タイマ

・機能

減算式 ON ディレー0.01ms 単位のタイマの動作をします。

(ラダー命令 TMUHX (557) と同様)

- ・実行条件が FALSE から TRUE になった時点で、タイマアドレスで指定したタイマを起動し、タイマ設定値で指定した値を元に現在値を 0.01ms おきに減算(-1)します。
- ・実行条件が TRUE の間、タイマ現在値の更新を実施し、現在値が 0 になると指定したタイマアドレスのタイムアップフラグが ON します。現在値が 0 でなければ、タイムアップフラグが OFF となります。
- ・実行条件が FALSE のときは、タイマアドレスの現在値に設定値をセットし、タイムアップフラグを OFF にします。

・使用方法

TMUHX(実行条件、タイマアドレス、タイマ設定値);

・引数/戻り値

変数名	データ型	内容
実行条件	BOOL	実行条件が ON の間タイマ動作が実行されます。
タイマアドレス	TIMER	使用するタイマアドレス(T0～T4095)変数を指定します。
タイマ設定値	UINT	ディレー時間を 0.01ms 単位で指定します。 (&0～&65535、#0～#FFFF)

・備考

- ・サイクルタイムが 10ms 以上のとき正常に動作しないことがあります。
- ・タイマ現在値をユーザプログラムで参照する場合には、タイミングによっては、実値と 1 サイクルタイム分のずれが生じることがあります。
- ・タイムアップフラグをユーザプログラムで参照する場合には、タイミングによっては、1 サイクルタイム分の遅れが生じることがあります。

● TTIMX: 積算タイマ

・機能

積算式 ON デイレー0.1s単位のタイマの動作をします。

(ラダー命令 TTIMX (555) と同様)

- ・実行条件が ON の間、現在値を加算(積算)します。
- ・実行条件が OFF になると、積算を停止し、現在値を保持します。

再度、実行条件が ON になったら、積算を再開します。

- ・タイマ現在値がタイマ設定値に到達するとタイムアップフラグが ON となります。
- ・タイムアップ後は、タイマ現在値およびタイムアップフラグの状態を維持します。
- ・リセット入力 that ON された場合に、タイマリセットを行います。

・使用方法

TTIMX(タイマ入力、リセット入力、タイマアドレス、タイマ設定値);

・引数

変数名	データ型	内容
実行条件	BOOL	実行条件が ON の間、現在値を加算（積算）します。
リセット入力	BOOL	リセット入力 that ON の場合に、タイマ現在値、タイムアップフラグがリセットされます。
タイマアドレス	TIMER	使用するタイマアドレス(T0～T4095)変数を指定します。
タイマ設定値	UINT	デイレー時間を 0.1s 単位で指定します。 (&0~&65535、#0~#FFFF)

・備考

- ・現在値の積算は命令実行時のみ実施するため、サイクルタイムが 100ms 以上の場合、正常に動作しないことがあります。
- ・タイムアップフラグをユーザプログラムで参照する場合には、タイミングによっては、1 サイクルタイム分の遅れが生じることがあります。

● CNTX: カウンタ

・機能

減算カウンタの動作をします。

(ラダー命令 CNTX (546) と同様)

- ・カウンタ入力の立上りごとにカウンタ現在値が減算カウントされます。
カウンタ現在値が 0 になると、カウントアップフラグが ON となります。
- ・リセット入力 that ON の時は、カウンタリセットとなり、現在値は設定値と等しくなります。
また、カウントアップフラグは OFF となり、カウント入力は無効となります。

・使用方法

CNTX(カウンタ入力、リセット入力、カウンタアドレス、カウンタ設定値);

・引数

変数名	データ型	内容
カウンタ入力	BOOL	カウンタ入力の立上りごとにカウンタ現在値が減算されます。
リセット入力	BOOL	リセット入力 ON の場合に、カウンタ現在値、カウントアップフラグがリセットされます。
カウンタアドレス	COUNTER	使用するカウンタアドレス(C0～C4095)変数を指定します。
カウンタ設定値	UINT	カウンタ減算開始のデフォルト値を指定します。 (&0～&65535、#0～#FFFF)

● CNTRX: 可逆カウンタ

・機能

加減算カウンタの動作をします。

(ラダー命令 CNTRX (548) と同様)

- ・加算カウンタ入力の立ち上がりで加算します。
- ・減算カウンタ入力の立ち上がりで減算します。
- ・加算により、現在値が設定値から 0 に桁上がりするとき、カウントアップフラグが ON となります。さらに 0 から 1 に加算するとカウントアップフラグが OFF となります。
- ・減算により現在値が 0 から設定値に桁下がりするとき、カウントアップフラグが ON となります。さらに設定値から 1 つ減算するとカウントアップフラグが OFF となります。
- ・立ち上がり入力、立下り入力同時 ON の場合は加減算しません。
- ・リセット入力 ON のとき現在値は 0 となり、カウンタ入力は無効となります。

・使用方法

CNTRX(加算カウンタ、減算カウンタ、リセット入力、カウンタアドレス、カウンタ設定値);

・引数

変数名	データ型	内容
加算カウンタ	BOOL	カウンタ入力の立上りごとにカウンタ現在値が加算されます。
減算カウンタ	BOOL	カウンタ入力の立上りごとにカウンタ現在値が減算されます。
リセット入力	BOOL	リセット入力 ON の場合に、カウンタ現在値、カウントアップフラグがリセットされます。
カウンタアドレス	COUNTER	使用するカウンタアドレス(C0～C4095)変数を指定します。
カウンタ設定値	UINT	カウンタ減算開始のデフォルト値を指定します。 (&0～&65535、#0～#FFFF)

● TRSET: タイマリセット

・機能

指定タイマをリセットします。

(ラダー命令 TRSET (549) と同様)

・使用方法

TRSET(実行条件、タイマアドレス);

・引数

変数名	データ型	内容
実行条件	BOOL	実行条件の ON の立ち上がりでタイマのリセットが実行されます。
タイマアドレス	TIMER	使用するタイマアドレス(T0～T4095)変数を指定します。

索引

A

AT 指定	2-10, 2-35
AT 指定（実アドレスへの割り当て）	1-20
AT 指定の方法	3-14
AutoGen	3-15

C

cxr	3-40
cxp	1-5

E

EN	2-24
----------	------

F

FB インスタンスエリア	1-22, 1-23, 2-16, 2-17, 2-18, 2-30, 3-27, 3-28
FB インスタンスメモリ	3-30
FB ライブラリファイル	3-6, 3-8
F キー	3-22

O

obj	1-5
-----------	-----

S

ST 言語使用時の制限	2-28
ST 言語の構成	5-9
ST 言語の仕様	5-2
ST ソースコード	6-12
ST 入力画面での表示	5-8
ST プログラム作成	6-6
ST プログラムのオンラインエディット	6-12
ST プログラムの記述上のルール	5-5
ST プログラムの作成	6-2
ST プログラムの挿入	6-2
ST プログラムの変数モニタ	6-10

あ

アドレスの内部割当	1-22
アルゴリズム	1-11
アルゴリズム作成	3-11
インスタンス	1-12
インスタンスの値のモニタ	3-45
インスタンスの仕様	2-16
インスタンスの生成	3-22
インスタンスの登録先	1-17
インスタンスの配置制限	2-28
インスタンスの呼出	2-23
インスタンス名称	2-16
インターロックの使用制限	2-26
インデックスレジスタ／データレジスタ	2-14, 2-25
インデックスレジスタの使用法	2-40
ウォッチウィンドウ	1-8, 3-44
エラーメッセージ一覧	4-3
オブジェクトコード	6-12
オムロン拡張関数	5-16
オンラインエディット	3-48, 6-12
オンラインエディットにおける制限事項	3-54, 6-14

か

外部と入出力の割付	3-24
外部変数（Externals）	1-19, 2-10, 2-34
カウンタ	2-18, 2-19
角度変換関数	5-17, 4-27
角度→ラジアン変換	4-27
クイックモード	3-51, 6-12
高機能ユニットの割付アドレスの指定方法	2-39
コンパイル	3-34

さ

出力パラメータ	1-13
出力変数（Outputs）	1-19, 2-6, 2-33
仕様	1-3, 4-3
初期値（Initial Value）	1-21, 2-14
数値文字列変更関数	5-13
制御文	5-20

た

代入文	5-19
タイマ	2-18, 2-19
タイマ系命令の動作	2-54
タイマ命令の使用制限	2-27
ダウンロード	3-41, 6-10
ダウンロードの制限	2-29
タスクへの割り付け	6-3
通信処理関数	4-19
データ型	2-10, 2-15, 5-3
データ型（Type）	1-20
データ型変更関数	5-13
デバッグ	3-42, 3-48, 6-10, 6-12
転送モード	3-51, 6-12
電断または運転開始時保持（Retained）	1-21, 2-14, 2-18

な

内部のプログラムを表示可能にする	3-8
内部変数（Internals）	1-19, 2-8, 2-35
内部割当アドレスの割り当てを最適化する	3-32
内部割当アドレスを確認する	3-30
入出力点数	2-49, 2-50, 2-51, 2-52, 2-53
入出力パラメータの設定	3-24
入出力変数	2-34
入出力変数（In Out）	1-19, 2-8, 2-34
入力パラメータ	1-13
入力変数（Inputs）	1-19, 2-6, 2-32
ネスティング	2-3, 2-24

は

配列型指定	2-36
配列指定	1-20, 2-11
配列指定の方法	3-14
パラメータ	1-9, 1-13
パラメータの仕様	2-22
パラメータへの値の入力方法	3-25
左からの文字列抽出	4-5
微分命令の使用制限	2-26
標準関数	4-5

標準モード	3-51, 6-12
ファイルメモリ用プログラムファイル	1-5
ファンクションブロック アドレス割付	3-27
ファンクションブロック メモリ情報	3-31
ファンクションブロック・ライブラリファイル	3-40
ファンクションブロック・ライブラリファイルの保存	3-40
ファンクションブロック・ライブラリファイルの読み出し	3-41
ファンクションブロック定義	1-8, 1-11
ファンクションブロック定義のオンラインエディット	3-48
ファンクションブロック定義のコピー	3-32
ファンクションブロック定義の新規作成	3-4
ファンクションブロック定義の生成	3-16
ファンクションブロック定義ファイル (*cxf)	1-26
ファンクションブロック定義名	2-4
ファンクションブロックとは	1-9
ファンクションブロックの作成	3-2
ファンクションブロックの仕様	2-2
ファンクションブロックのプロパティ	3-8
ファンクションブロック呼び出しの更新	3-24
ファンクションブロックをファイルからロード	3-41
ファンクションブロックをファイルへ保存	3-40
複数のインスタンス化	2-21
プログラミング	1-10
プログラミングコンソール	2-29
プログラムのプロパティ	6-3
プログラムのモニタ	3-44
プロジェクトファイル	1-5
変数	1-18
変数種類の決め方	2-32
変数定義	1-11, 2-4
変数の種類	1-19, 2-5
変数の属性	1-20, 2-10
変数の属性と変数種類の関係	1-21
変数のデータ型の決め方	2-32
変数の登録	3-9, 3-13, 3-15
変数の内部割当範囲設定	3-27
変数名	2-10
変数名 (Name)	1-20
変数名を自動的に生成する	3-15

ま

右からの文字列抽出	4-5, 4-9, 4-10, 4-12, 4-13, 4-14, 4-15, 4-28, 4-29, 4-30, 4-31, 4-32
命令	2-25
メインメニュー	1-6, 1-7, 1-8
メモリカード処理関数	5-16, 4-17
文字列関数	4-5
文字列長の検出	4-5
文字列の結合	4-6
文字列の検索	4-8
文字列の削除	4-7
文字列の受信 (CPU ユニットの RS-232C ポート)	4-23
文字列の受信 (シリアルコミュニケーションボード (SCB) のシリアルポート)	4-24
文字列の受信 (シリアルコミュニケーションユニット (SCU) のシリアルポート)	4-25
文字列の送信 (CPU ユニットの RS-232C ポート)	4-19

文字列の送信 (シリアルコミュニケーションボード (SCB) のシリアルポート)	4-20
文字列の送信 (シリアルコミュニケーションユニット (SCU) のシリアルポート)	4-21
文字列の置換	4-7
文字列の抽出	4-6
や	
予約メモリの確保方法	3-48
ら	
ライブラリ化	1-24
ライブラリの作成	1-10
ラジアン→角度変換	4-27
ラダー/ST 言語の共通の制限	2-28
ラダー言語使用時の命令制限	2-25
ラダープログラムの導通モニタ	3-42
ログファイルの作成	4-17

オムロン株式会社 インダストリアルオートメーションビジネスカンパニー

製品に関するお問い合わせ先



フリー
通話 **0120-919-066**
携帯電話・IP電話などではご利用いただけ
ませんので、右記の電話番号へおかけください。
055-982-5015
(通話料がかかります)
受付時間：9:00～19:00 (12/31～1/3を除く)



オムロンFAクイックチャット

www.fa.omron.co.jp/contact/tech/chat/

技術相談員にチャットでお問い合わせいただけます。(I-Webメンバーズ限定)

受付時間：平日9:00～12:00 / 13:00～17:00 (土日祝日・年末年始・当社休業日を除く)

※受付時間、営業日は変更の可能性がございます。最新情報はリンク先をご確認ください。



その他のお問い合わせ：納期・価格・サンプル・仕様書は貴社のお取引先、または貴社担当オムロン販売員にご相談ください。オムロン制御機器販売店やオムロン販売拠点は、Webページでご案内しています。



オムロン制御機器の最新情報をご覧ください。緊急時のご購入にもご利用ください。 www.fa.omron.co.jp

- 本誌に記載のない条件や環境での使用、および原子力制御・鉄道・航空・車両・燃焼装置・医療機器・娯楽機械・安全機器、その他人命や財産に大きな影響が予測されるなど、特に安全性が要求される用途に使用される際には、当社の意図した特別な商品用途の場合や特別の合意がある場合を除き、当社は当社商品に対して一切保証をいたしません。
- 本製品の内、外国為替及び外国貿易法に定める輸出許可、承認対象貨物(又は技術)に該当するものを輸出(又は非居住者に提供)する場合は同法に基づく輸出許可、承認(又は役務取引許可)が必要です。

オムロン商品のご用命は